INTERACTIVE GRAPH REDUCTION AND ANALYSIS PROGRAM.

James Winton Thomas

United States
Naval Postgraduate School

THESIS

INTERACTIVE GRAPH REDUCTION AND ANALYSIS PROGRAM

by

James Winton Thomas

June 1970

This document has been approved for public release and sale; its distribution is unlimited.

Interactive Graph Reduction and Analysis Program

by

James Winton Thomas
Lieutenant (junior grade), United States Navy
B.S., Tufts University, 1969

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1970

ABSTRACT

This paper describes the design of an interactive system to aid in the analysis of problems which involve directed graphs. The digital computing system is assumed to have a graphic display device on which directed graphs may be drawn and from which light pen, function keyboard, and alphanumeric keyboard information may be transmitted on-line to the system. Directed graphs are represented in core storage by a dynamically allocated hierarchical list structure. User-written analysis routines are linked to the system to apply it to a particular field of problems. An initial implementation of its capabilities on the IBM 360/67 with an IBM 2250 Display Unit was written in PL/I (F). Under the IBM System/360 Operating System, it executed in less than 200K bytes and provided reasonable response to on-line interaction.

TABLE OF CONTENTS

I. INTRODUCTION

This paper describes a system designed to aid in the on-line analysis of problems which involve directed graphs. Likely applications include PERT network analysis, electronic circuit analysis and artificial intelligence. A PL/I (F) program was implemented on the IBM System/360 Model 67 using the IBM 2250 Display Unit and approximately 200K bytes of core storage. The program demonstrates some of the capabilities of the system described and is appended with a users' manual to this paper.

Chapter II gives definitions and explains concepts used in the paper and the program. Directed graphs, progressions, reachable sets, and cells and pointers used in list processing are presented.

The design of the program is described in Chapter III. Discussed therein are some of the goals of an interactive system. This chapter also contains a detailed description of a hierarchical list structure representing a directed graph and a sample algorithm which traverses the structure.

Chapter IV concludes the paper with some comments on the system and implementation. A program listing appears as the last section of the paper. Output from a sample run is also included with some pictures of the 2250 Display Unit during a run. Finally, Appendix A contains a users' manual for the program.

II. FUNDAMENTAL CONCEPTS

In this chapter, basic concepts of directed graphs essential to understanding this paper are developed. Many of the definitions are from a text written by Busacker and Saaty [1]. Their textbook Finite Graphs and Networks contains concise definitions used in graph theory. Their book also discusses many graph theory applications.

Some definitions in graph theory have not found universal acceptance. In order to avoid unnecessary confusion, this paper will use the terminology consistent with Busacker and Saaty [1].

A directed graph is usually an abstract model of a real situation. To arrive at an unambiguous definition of an abstract graph, Euclidean 3-dimensional space is employed to define a geometric graph, then from that the definition of an abstract graph is obtained. Euclidean 3-space consists of triplets of real numbers.

A. GEOMETRIC GRAPHS

A simple open curve in Euclidean 3-space is a continuous, non-self-intersecting curve joining two distinct points. A simple closed curve is the same as a simple open curve except that its endpoints coincide. Now, a geometric graph in Euclidean 3-space is a set $V = \{v_i\}$ of points in Euclidean 3-space and a set $A = \{a_i\}$ of simple curves (open and closed) satisfying the following conditions:

1) The closed curves in A contain only one point in V.

2) The open curves in A have as endpoints two points in V.

3) No two curves in A have common points except for points in V.

Thus, a geometric graph is simply a system of points connected by curves. The curves intersect one another only at points in V. The most common real life example might be a network of highways--intersections representing points and the highways between intersections representing curves.

B. ABSTRACT GRAPHS

The notion of a geometric graph is indeed concise, but carries with it more information than is normally needed in the applications of graph theory. The exact spacial location of the points of a graph in Euclidean space is not essential in most directed graph analysis. The most important property of a geometric graph is the connectivity of the graph: which two vertices in V are connected by a curve in A. The connectivity of a graph now can be described with the following notation:

$$\Delta(a_i) \longrightarrow (v_j, v_k)$$

which reads, "curve a_i starts at vertex v_j and ends at vertex v_k". In this manner each curve in A is mapped into an ordered pair of vertices in V. Each of the possible ordered pairs of the elements in V are elements of the cartesian cross product V X V. If $V = \{v_1, v_2\}$, then the set $V \times V = \{(v_1,v_1), (v_1,v_2), (v_2,v_1), (v_2,v_2)\}$.

An <u>abstract graph</u> or simply a <u>graph</u> is composed of a set (of vertices) V, a set (of arcs) A, and a mapping of A into V X V. The definition must also include the restriction that no element of A is in V. This last statement stresses the point that in an abstract graph a point cannot also be a curve.

The system $G_1 = (V_1, A_1, \Delta_1)$ is a subgraph of a graph $G = (V, A, \Delta)$ if the following conditions are satisfied:

1) V_1 is a subset of V and A_1 is a subset of A.

2) For every a in A_1, $\Delta_1(a) = \Delta(a)$.

3) For every a in A_1, if $\Delta_1(a) \dashrightarrow (v_i, v_j)$, then v_i and v_j are elements of V_1.

The system $G_1 = (V_1, A_1, \Delta_1)$ is an induced subgraph of $G = (V, A, \Delta)$ if the following conditions are satisfied:

1) G_1 is a subgraph of G.

2) Every arc a, $\Delta(a) \longrightarrow (u,v)$, in A is in A_1 if u and v are elements of V_1.

Although an abstract graph is not defined in a 3-dimensional space, it is often useful if not invaluable to picture this abstract model by drawing it on a piece of paper or displaying it on a cathode ray tube. Through man's heuristic properties, drawings and pictures seem to mean more than just describing a graph as two sets and a mapping function. Solving some graph theory problems depends to some degree on a drawing of a graph. Any drawing of a graph, for which the only common points of arcs are the vertices, is called a geometric realization of the graph. Many geometric realizations of a given graph exist. If a graph is defined as $G = (V, A, \Delta)$, $V = \{v_1, v_2, v_3\}$, $A = \{a_1, a_2, a_3\}$, and $\Delta(a_1) \longrightarrow (v_1, v_2)$, $\Delta(a_2) \longrightarrow (v_2, v_3)$, and $\Delta(a_3) \longrightarrow (v_1, v_3)$; then one comprehends more about the graph by looking at some of its geometric realizations shown in Figure 1. Each one, because it looks different from the others, imparts different unquantifiable information. This is one of the reasons for the use of the IBM 2250 Display Unit in the design and operation of the system.

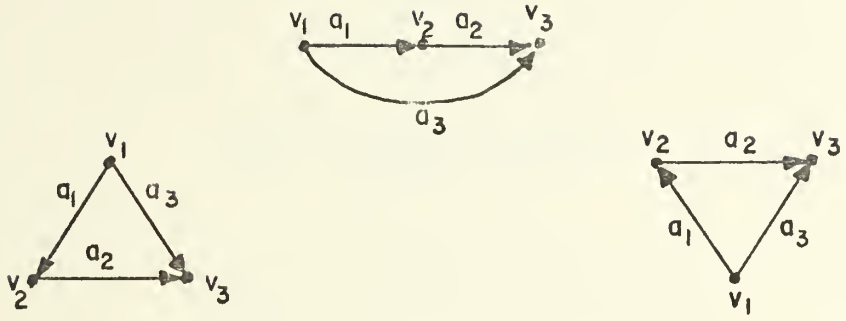


Figure 1. Three geometric realizations of a graph

C. PROGRESSIONS

An arc progression is an ordered sequence of arcs $a_1, a_2, a_3, \ldots, a_n$ satisfying one condition:

An ordered sequence of nodes $v_1, v_2, \ldots, v_{n+1}$ must exist such that

$$\Delta(a_i) \rightarrow (v_i, v_{i+1}), \quad i = 1,2,3, \ldots, n.$$

A path progression is an arc progression for which $v_1 \neq v_{n+1}$, and no arc appears in the sequence more than once.

A cycle progression is an arc progression for which $v_1 = v_{n+1}$ and no arc appears in the sequence more than once.

The length of a progression is simply the number of arcs in the sequence. A progression is said to lead from v_1 to v_{n+1}, or, the progression starts at v_1 and ends at v_{n+1}. An arc a which maps into (v,w) is positively incident with the initial node v and negatively incident with the terminal node w.

Consider the graph G in Figure 2. Some progressions, their type, length, and starting and ending nodes are shown in Table I. Some sequences of arcs which are not progressions are shown in Table II.

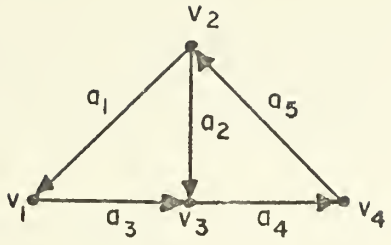


Figure 2. A directed graph

Some progressions	Type	Length	From	To
a_1	path	1	v_2	v_1
a_3,a_4	path	2	v_1	v_4
a_2,a_4,a_5	cycle	3	v_2	v_2
a_2,a_4,a_5,a_2	arc	4	v_2	v_3
a_1,a_3,a_4,a_5	cycle	4	v_2	v_2
a_4,a_5,a_1,a_3	cycle	4	v_3	v_3

Table I. Progressions in graph G in Figure 2.

a_1,a_2	a_5,a_2,a_1	a_2,a_3
a_4,a_3	a_3,a_2,a_4	a_1,a_3,a_2

Table II. Some sequences in graph G which are not progressions.

D. REACHABLE SETS

During the analysis of a graph, some algorithms "look at" a node, say v_i, and ask, "Which nodes are at the end of arcs starting at v_i?" Using the definition of a progression, the question may be restated: Which nodes are at the end of all length 1 progressions starting at v_i? This set of nodes is called the reachable set of the node v_i and is denoted $R(v_i)$. The superscript of the reachable set is the length of progressions considered. $R^n(v_i)$ is the set of nodes which are at the end of all length n progressions starting at v_i.

$R^{-n}(v_i)$ is the set of starting nodes of all length n progressions ending at v_i. $R^n(v_1,v_2,\ldots,v_j)$ is the union of the sets $R^n(v_1)$, $R^n(v_2)$, ..., $R^n(v_j)$. To cover all possible integer values of n, $R^0(v_i) = \{v_i\}$.

An arc is said to be a loop is it starts and ends at the same node. Therefore, if a_i is a loop starting and ending at v_i, then v_i is in $R(v_i)$. Further, since an arc may be repeated in an arc progression, v_i would also be in $R^n(v_i)$ for all values of n.

Consider the graph G in Figure 3. Some reachable sets are presented in Table III. Note that if $R^n(v_i)$ of any node v_i contains v_2, then $R^{n+m}(v_i)$ also contains v_2 for m = 1, 2, 3, ... due to the loop a_6.

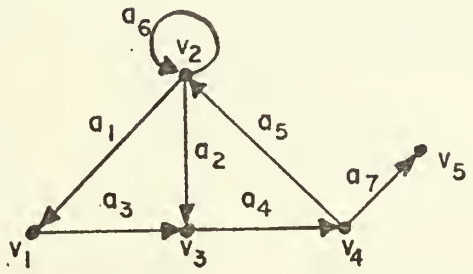


Figure 3. A directed graph with a loop

n	v	$R^n(v)$	arc progressions
1	v_2	v_2,v_1,v_3	a_6; a_1; and a_2
2	v_4	v_2,v_1,v_3	a_5,a_6; a_5,a_1; and a_5,a_2
3	v_1	v_2,v_5	a_3,a_4,a_5 and a_3,a_4,a_7
6	v_3	v_2,v_1,v_3,v_4,v_5	a_4,a_5,a_6,a_6,a_6,a_6; a_4,a_5,a_6,a_6,a_6,a_1; a_4,a_5,a_2,a_4,a_5,a_2; a_4,a_5,a_6,a_6,a_2,a_4; and a_4,a_5,a_6,a_2,a_4,a_5
-2	v_4	v_2,v_1	a_2,a_4 and a_3,a_4
-3	v_5	v_1,v_2	a_2,a_4,a_7 and a_3,a_4,a_7
2	$R^3(v_1)$	v_2,v_1,v_3,v_4	a_6,a_6; a_6,a_1; a_1,a_3; and a_2,a_4

Table III. Some Reachable Sets of graph G in Figure 3.

The definition of a path reachable set is similar to that of the reachable set. The path reachable set of node v_i, $P^n(v_i)$, is the set of all nodes at the end of all length n path progressions starting at v_i. $P^{-n}(v_i)$ is defined as the set of starting nodes of all length n progressions ending at v_i. Again, to cover all possible integer values of n, $P^0(v_i)$ is defined as the empty set φ. $P^n(v_1,v_2,v_3,\ldots,v_k) = \left\{ \bigcup_{i=1}^{k} P^n(v_i) \right\}$.

A cumulative reachable set $R_c^n(v_j) = \left\{ \bigcup_{i=0}^{n} R^i(v_j) \right\}$. Likewise, the cumulative path reachable set $P_c^n(v_j) = \left\{ \bigcup_{i=1}^{n} P^i(v_j) \right\}$. Table IV shows some path reachable sets of graph G in Figure 3.

n	v	$P^n(v)$	path progressions
1	v_2	v_1,v_3	a_1 and a_2
2	v_4	v_3,v_1,v_2	a_5,a_2; a_5,a_1 and a_5,a_6
3	v_1	v_2,v_5	a_3,a_4,a_5 and a_3,a_4,a_7
4	v_3	v_1	a_4,a_5,a_6,a_1
-2	v_4	v_1,v_2	a_3,a_4 and a_2,a_4
-3	v_5	v_1,v_2	a_3,a_4,a_7 and a_2,a_4,a_7
2	$P^3(v_1)$	v_1,v_3,v_4	a_6,a_1; a_1,a_3; and a_2,a_4

Table IV. Some path reachable sets of graph G in Figure 3.

E. SUPER-NODES AND SUPER-ARCS

In many applications of graph theory, the number of nodes and arcs is quite large. Consider the geometric realization of a graph of every street, alley, thoroughfare and intersection of a large city like Chicago. If one could find a sheet of paper large enough on which to draw it, the probability of drawing it without error would be quite small. Assuming that it has been drawn successfully, using it to answer various questions about Chicago would be difficult if not impossible! For instance, if the question, "How many thoroughfares

intersect Interstate 80," were asked, one would have to sift through the maze of alleys and residential streets to first locate Interstate 80 at some point on the drawing. Then one would have to follow it asking at each intersection if the streets connected to Interstate 80 were thoroughfares. It would be much easier to attack the overall problem in a manner similar to the approach used by the county cartographers. Draw a map showing just Interstate highways and intersections with major thoroughfares. Also put on the map references to sectional maps showing in a little more detail the area concerned. The sectional map might also contain references to detail maps showing all streets and alleys of a sub-subsection of the city. Answering the questions above would be much easier. One would not be forced to bother with the small streets and alleys. If a question were asked which called for information about the small streets and alleys of the city, one could reference the detail maps as desired.

This is perhaps an overworked example justifying the concept of a super-node. The idea is simple: Represent an induced subgraph with a single super-node, maintaining connections to and from the subgraph in the following manner.

Let $G_1 = (V_1, A_1, \Delta_1)$ be an induced subgraph of $G = (V, A, \Delta)$. Let super-node v_{11} be an element of V representing the subgraph G_1.

If $\Delta(a) \longrightarrow (u,v)$ such that $u \in V\text{-}V_1$ and $v \in V_1$, then a is negatively incident with v_{11}.

If $\Delta(a) \longrightarrow (v,u)$ such that $v \in V_1$ and $u \in V\text{-}V_1$, then a is positively incident with v_{11}.

An arc which is incident with a super-node is called a <u>super-arc</u>.

To completely specify a graph containing super-nodes and super-arcs, additional notation is used to define a graph $G = (V, A, \nabla)$. To completely

specify a super-node in V, the graph that it represents must be associated with the super-node. If v is a super-node in V representing graph $G_1 = (V_1, A_1, \nabla_1)$, then the node v appears with the name of the represented graph enclosed in parentheses. Thus, $v(G_1)$ is an element of V. The set of arcs A is the same as defined before. a_i is an element of A if a_i is a simple arc or super-arc in G. Additional notation ∇ replacing the mapping Δ indicates all incident nodes of all arcs.

The sequence of characters $\nabla(a)$ is defined for each arc a. The characters consist of parentheses, brackets, commas and node names. The order of characters is determined in the following manner:

$\nabla(a)$, the sequence of characters defining arc a in G, is defined using an ordered pair of sequences.

$$\nabla(a) = ([F_a(u)], [F_a(v)]).$$

where

u is the starting node in G of arc a, and
v is the ending node in G of arc a.

The sequence of characters represented by $F_a(v)$ is defined recursively:

$$F_a(v) = \begin{cases} v & \text{if v is a simple node.} \\ v, F_a(v') & \text{if v is a super-node.} \end{cases}$$

If v is a starting node of arc a, then v' is the starting node of a in the graph represented by v.

If v is an ending node of arc a, then v' is the ending node of a in the graph represented by v.

The recursion must give a finite sequence of nodes, because every arc eventually starts or ends at a simple node.

If $\nabla(a) = ([u_1, u_2, u_3, \ldots, u_i], [v_1, v_2, v_3, \ldots, v_j])$, then a, u_1, and v_1 are in the same graph. u_1 and v_1 are the starting and ending

super-nodes of the super-arc a. u_i and v_j are the starting and ending simple nodes of arc a. u_k and v_t are in the subgraphs represented by u_{k-1} and v_{t-1} for k = 2,3,4, ..., i and t = 2,3,4,...,j.

The graphs shown in Figures 4, 5, and 6 demonstrate the use of the super-node concept. First, a graph G is defined in Figure 4.

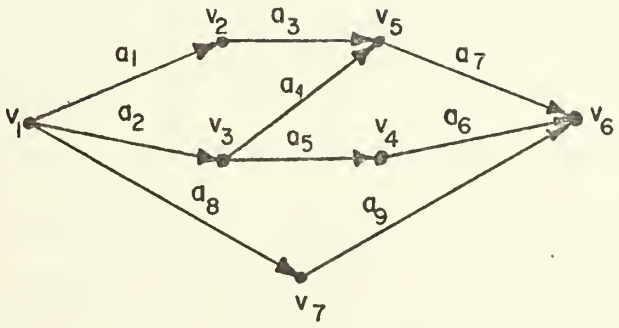
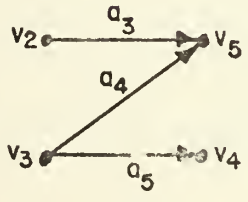
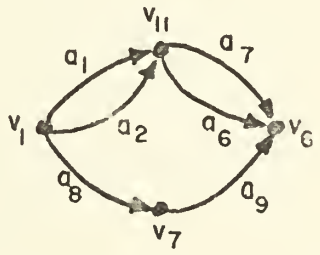


Figure 4. A detailed graph

Then, replacing the induced subgraph determined by v_2, v_3, v_4, and v_5 with a super-node v_{11} gives the graphs G_1, the detail graph represented by super-node v_{11}, and Gi, the altered graph G. Arcs a_3, a_4 and a_5 are part of the subgraph G_1. Graphs G' and G_1 are shown in Figure 5.



(a) Graph G_1



(b) Graph G'

Figure 5. A graph G' containing a super-node

The following information not shown about the super-arcs is maintained in the sequences $\nabla(a)$:

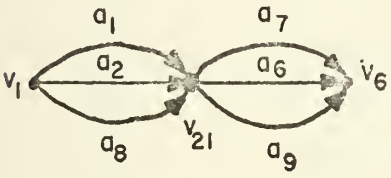
Super-arc a_1 not only ends at v_{11} in G', but also at v_2 in G_1.

In a similar manner, arc a_2 ends at v_{11} in G' and at v_3 in G_1.

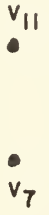
Super-arc a_6 starts at v_{11} in G' and v_4 in G_1.

Super-arc a_7 starts at v_{11} in G' and v_5 in G_1.

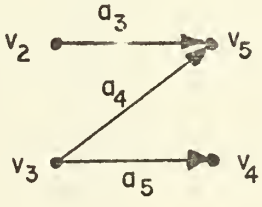
Noting that v_{11} and v_7 are the ending nodes of two arcs which start at v_1, and are also the starting nodes of two arcs which end at v_6; graph G" is constructed by representing with super-node v_{21} the induced subgraph G_2 consisting of v_{11} and v_7. Since all arcs and super-arcs of G' start or end at v_{11} or v_7, they all become super-arcs in G". The resulting graphs are shown in Figure 6.



(a) Graph G"
v_{21} represents G_2



(b) Graph G_2
v_{11} represents G_1



(c) Graph G_1

Figure 6. A system of graphs containing super-nodes and super-arcs

The following information not shown is maintained in the (a) sequences:

Super-arc a_1 ends at v_{21} in G'', v_{11} in G_2 and v_2 in G_1.

Super-arc a_2 ends at v_{21} in G'', v_{11} in G_2 and v_3 in G_1.

Super-arc a_8 ends at v_{21} in G'' and v_7 in G_2.

Super-arc a_7 starts at v_{21} in G'', v_{11} in G_2 and v_5 in G_1.

Super-arc a_6 starts at v_{21} in G'', v_{11} in G_2 and v_4 in G_1.

Super-arc a_9 starts at v_{21} in G'' and v_7 in G_2.

Table V(a) shows the sequences $F_a(v)$ for the system of graphs in Figure 6. The sequences $\nabla(a)$ of the system are in Table V(b). These sequences contain all information necessary to construct the original graph G in Figure 4.

v	a	$F_a(v)$
v_1	a_1, a_2, a_3, a_6, a_7, and a_9	v_1
v_2	a_1 and a_3	v_2
v_3	a_2, a_4 and a_5	v_3
v_4	a_5 and a_6	v_4
v_5	a_3, a_4 and a_7	v_5
v_6	a_6, a_7 and a_9	v_6
v_7	a_8 and a_9	v_7
v_{11}	a_1	$v_{11}, F_{a1}(v_2) = v_{11}, v_2$
v_{11}	a_2	$v_{11}, F_{a2}(v_3) = v_{11}, v_3$
v_{11}	a_6	$v_{11}, F_{a6}(v_4) = v_{11}, v_4$
v_{11}	a_7	$v_{11}, F_{a7}(v_5) = v_{11}, v_5$
v_{21}	a_1	$v_{21}, F_{a1}(v_{11}) = v_{21}, v_{11}, v_2$
v_{21}	a_2	$v_{21}, F_{a2}(v_{11}) = v_{21}, v_{11}, v_3$
v_{21}	a_6	$v_{21}, F_{a6}(v_{11}) = v_{21}, v_{11}, v_4$
v_{21}	a_7	$v_{21}, F_{a7}(v_{11}) = v_{21}, v_{11}, v_5$
v_{21}	a_8	$v_{21}, F_{a8}(v_7) = v_{21}, v_7$
v_{21}	a_9	$v_{21}, F_{a9}(v_7) = v_{21}, v_7$

(a) The sequences $F_a(v)$ for the graphs

a	$\nabla(a)$
a_1	$([v_1], [v_{21},v_{11},v_2])$
a_2	$([v_1], [v_{21},v_{11},v_3])$
a_3	$([v_2], [v_5])$
a_4	$([v_3], [v_5])$
a_5	$([v_3], [v_4])$
a_6	$([v_{21},v_{11},v_4], [v_6])$
a_7	$([v_{21},v_{11},v_5], [v_6])$
a_8	$([v_1], [v_{21},v_7])$
a_9	$([v_{21},v_7], [v_6])$

(b) The sequences $\nabla(a)$ for the graphs

Table V. $F_a(v)$ and $\nabla(a)$ for the graphs in Figure 6.

F. CELLS AND POINTERS

The internal representation of the user's graphs are in the form of a list which is composed of cells. A cell is a logically contiguous block of computer storage locations. The storage locations in a cell are divided into one or more fields. Each cell has a unique address which is the storage location of the first field in the cell. Each field of a cell has a unique displacement from the first field. The address of a field of a particular cell is the address of the cell plus the displacement of the field within the cell. A field may be used to store characters, real or integer numbers, or addresses of other cells. A storage location (not necessarily in any cell) is called a pointer if it is used to hold the address of a cell. Thus, a cell may contain fields used as pointers (i.e., a cell may point to several other cells or itself!). A unique pointer value is reserved to indicate that the pointer with this value does not point to a cell. This value is called the NULL value and is represented by the character

Λ. A variable is a symbolic name representing the contents of a unique storage location not in any cell.

Consider a list representing a deck of cards. Each cell represents one card. Each cell consists of three fields named DENOM, SUIT, and NEXT. DENOM is the denomination of the card (2, 3, ... king, ace), SUIT is the suit (club, heart, etc.) and NEXT is a pointer identifying the cell which represents the next card in the deck. TOP, LAST and P are unique pointer variables. To aid in interpreting the list in Figure 7b, a key is given, Figure 7a, to show the relative position of the fields in a drawing of a CARD cell. The values of pointers are indicated by arrows. Note that the NEXT arrows start in a cell and point to the boundary of the next cell.

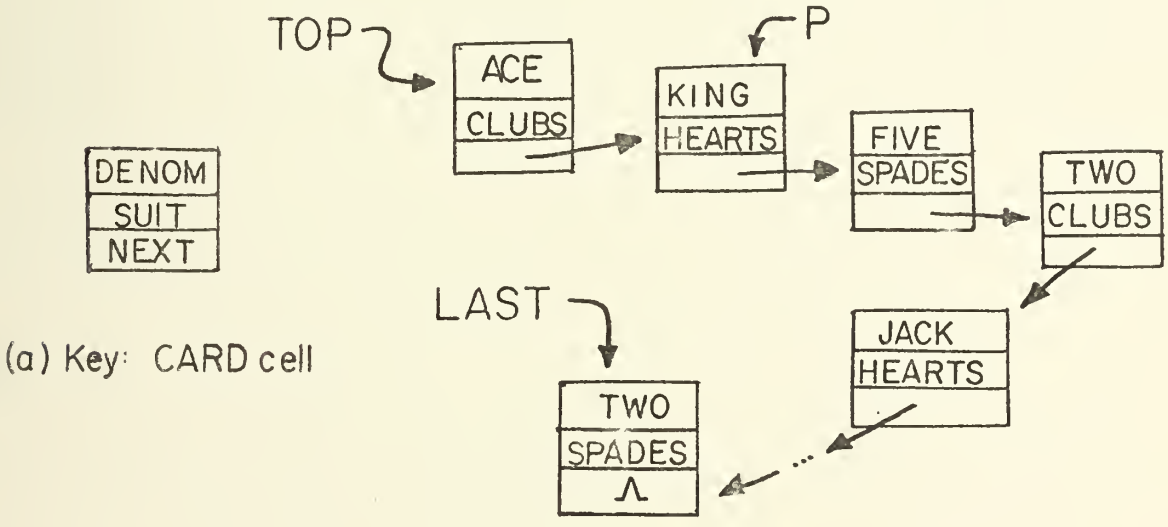


Figure 7. A list representing a card deck

TOP is a pointer variable which identifies the top card of the deck. TOP's DENOM is "ace". We may set P (another pointer variable) equal to TOP's NEXT as shown. Then P's DENOM is "king". P may be moved on down the deck by setting P equal to P's NEXT. Finally, P will equal LAST. Note that LAST's NEXT is NULL indicating the end of the deck.

One need not be concerned with assigning particular storage locations to variables and cells. Higher level programming language compilers accomplish this clerical task. The important point is that each variable is a different storage location not among those comprising cells.

III. THE DESCRIPTION OF THE PROGRAM

A. GOALS OF THE PROGRAM

As it is implied by the title, all of the goals of the system may be summed up as follows: To provide a man-machine interactive system used in solving directed graph problems.

To be truly interactive, the total system (hardware and software) must keep the user's mind on the directed graph problem. Use of a computing system with a cathode ray tube display unit allows instantaneous communications between the user and a program. Also, the program must be efficient; the system must process the commands as fast as the user can issue them. Indeed, the user realizes that an on-line system is to be responsive and becomes irritated when it is not. Peak loads on a multiprogrammed computing system will cause a lag in response time no matter how efficient the software is. Since the on-line user becomes irritated upon waiting four seconds or more for the machine to avail itself for the next command, the on-line system should frequently test the time elapsed from the entry of the last user command. If the elapsed time is approaching four seconds, processing should be interrupted to display an apologetic message to the user explaining the cause of delay.

It has been observed that users of on-line systems become irritated when obviously needed commands are not available or selection of commands is a complicated process (e.g., searching a list of operations for a numeric code to be entered). The user's train of thought is forced to leave the realm of the problem to become involved with the inadequacies

of the system. Therefore, this system is designed to use the light pen detection facility and a complete list of alphabetic mnemonic commands displayed on the screen of the display unit--the simplest and quickest means available for communications between the system and the user.

The on-line user realizes that the system is merely a device existing to serve him; therefore, he should not be treated rudely by the system. If he is, he will choose not to use the system, or if forced to, will spend many an unhappy hour with it--definitely not a characteristic of a good interactive system. The tone of messages to the user incorporated in this system is that of a courteous laborer working for the user.

Conserving the resources of the user should be a goal of any system, on-line or not. It is assumed that the multiprogrammable computer will have three or more jobs to work on while not processing commands from the user. Thus, the facilities of the computer are not wasted by having it wait for the user to enter a command. Also, use of a program overlay feature decreases the amount of core storage allocated to this system.

This system is designed to use dynamic storage allocation instructions to construct a PL/I list structure representing the user's graphs. The use of this feature not only conserves storage, but lends the use of the system to a variety of graph theory applications. User-written routines are linked to the system thus specializing it for use with a particular application. The user, through the routines he provides, may associate additional fields with nodes and arcs (e.g., resistance, inductance, etc.). He also provides analysis routines which access the

list structures. System routines are provided to aid the user in his analysis and display of his results.

Many short sessions with a system is psychologically more pleasant than fewer extended sessions. Therefore, directed graphs may be saved as a sequential data set at the end of a session. This data set may be used to initialize the system at the beginning of the next session.

Because of the interactive nature of the system, it lends itself to on-line design of directed graphs. Thus, the user may modify the graph, call his analysis routines and from the output of the analysis, make more modifications, and so forth until the optimal design is reached. He may prepare before his first session a card deck from which the system will obtain the initial graph structure. The card format is designed to be relatively free of restrictions. To fully utilize the interactivity of the system, corrections for errors found in the card deck are requested from the user on-line.

In conclusion, the system designed is flexible--to meet the desires of a variety of users. It is on-line--to take advantage of man's heuristic nature. And, it is courteous--to make use of the system pleasant.

B. SYSTEM DESIGN

Because the user must program his own analysis routines and link them to the system, a discussion of the overall design is presented.

Basically, the user is provided with a PL/I main program and a series of subroutines which perform three functions:

1) Drive the display unit.

2) Generate and modify the user's list-structured directed graphs.

3) Call user analysis routines.

When a function key at the display unit is pressed, the system calls the user routine named BUTTONS. Since the system variables are declared "EXTERNAL," the user has access to their current values and can use them in his analysis. The user may display the results of his analysis by calling a system routine to save the current display preparing it for his use; or, by calling call another routine which displays a "page" of user-generated text. Use of either procedure allows him to use the light pen, function keyboard, and alphanumeric keyboard for his own purposes.

C. THE LIST STRUCTURE

The PL/I List Processing features are used extensively to construct a representation of the user's directed graphs in core memory. Allocations of two based structures which represent two types of cells are linked together to form the directed graphs. NODE is the level 1 identifier of the structure used to represent nodes of the graph and ARC is the level 1 identifier of the structure used to represent an arc. Each allocation of either structure is called a cell. The structures are declared as follows:

```
DECLARE  1 NODE BASED (PP),
           2 LABEL CHARACTER(4),
           .
           2 NEXT POINTER,
           .
           .
           2 DOWN POINTER,

         1 ARC BASED (P-ARC),
           2 SON POINTER,
           .
           .
           2 MORE POINTER;
```

To avoid confusion, some of the level 2 elements in the structures are not shown here. They will be explained later.

The integer 1 preceding NODE and ARC indicate that NODE and ARC are symbolic names representing all of the fields of one of the cells. The integer 2 beginning the lines under 1 NODE and 1 ARC establish symbolic names and displacements for the fields in the NODE; likewise for SON and MORE in the ARC cells. The word BASED instructs the PL/I compiler to generate code which will dynamically allocate core storage upon the execution of the ALLOCATE statement and will free storage upon the execution of the FREE statement.

The NEXT and DOWN fields in NODE cells and SON and MORE in ARC cells are used as pointers, that is, they are used to hold addresses of unique cells. By program convention, the DOWN and SON fields always point to ARC cells. This makes traversing the lists less complicated.

A NODE cell is allocated for each node and an ARC cell is allocated for each arc of a graph. The DOWN pointer of the NODE cell points to a list of ARC cells. The MORE field links the list of ARC cells similar to the way NEXT linked the CARD cells in Figure 7. All of the arcs represented by the ARC cells in a list start at the same node. The SON pointer of an ARC cell points to the ending node of the arc. Figure 8 shows a list structure demonstrating the use of the two fields in each cell. It is called a partial list structure because not all fields are shown. The letters appearing immediately above each ARC cell serve merely to identify the ARC cell; only node labels are stored in the list structure.

From Figure 8, we may make the following observations:

1) ARC cell A1 represents arc a_1.
   ARC cell A2 represents arc a_2.
   ARC cell A3 represents arc a_3.

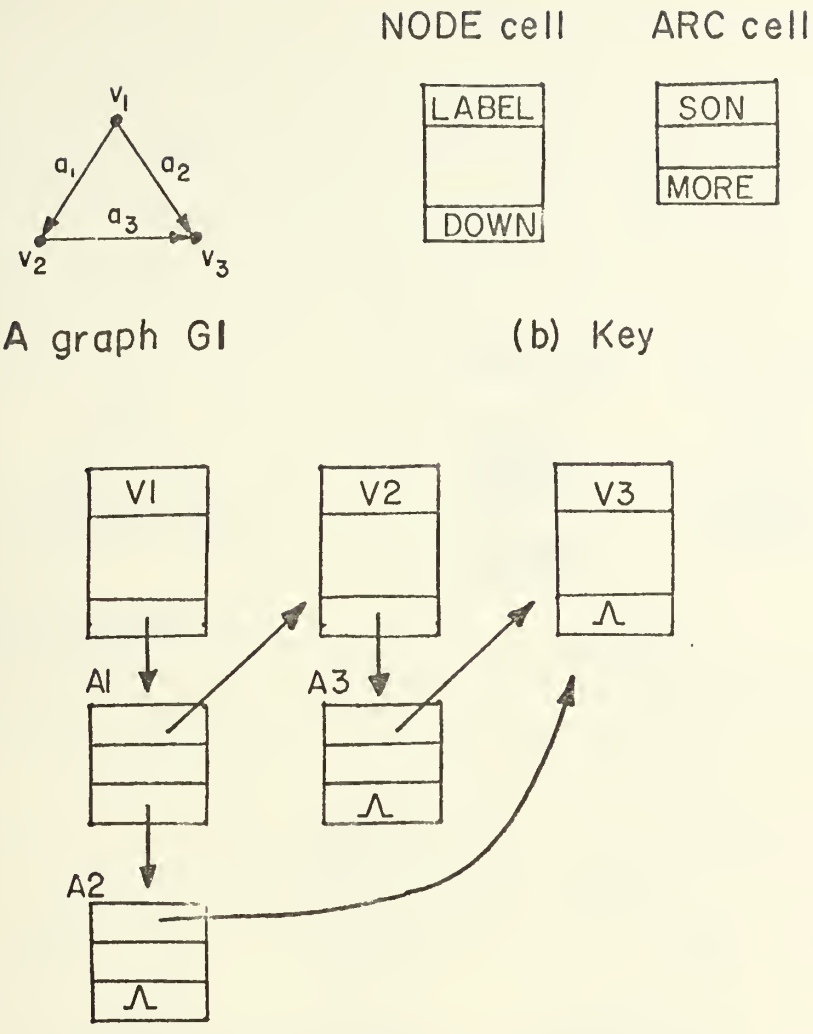


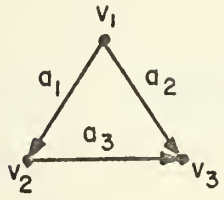
Figure 8. A graph GI and its list structure

2) The label of a node is stored in the LABEL field of the NODE cells.

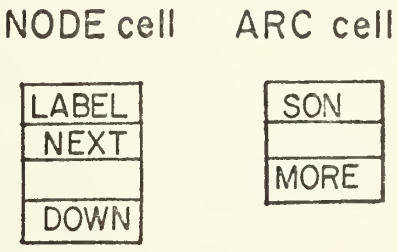
3) The DOWN pointer of NODE cell labelled V3 has a NULL value indicating that there are no arcs starting at v_3.

4) There is no unique order for arcs appearing on the list of ARC cells. For instance, ARC cell A2 may have come directly under NODE cell V1 and ARC cell A1 under ARC cell A2.

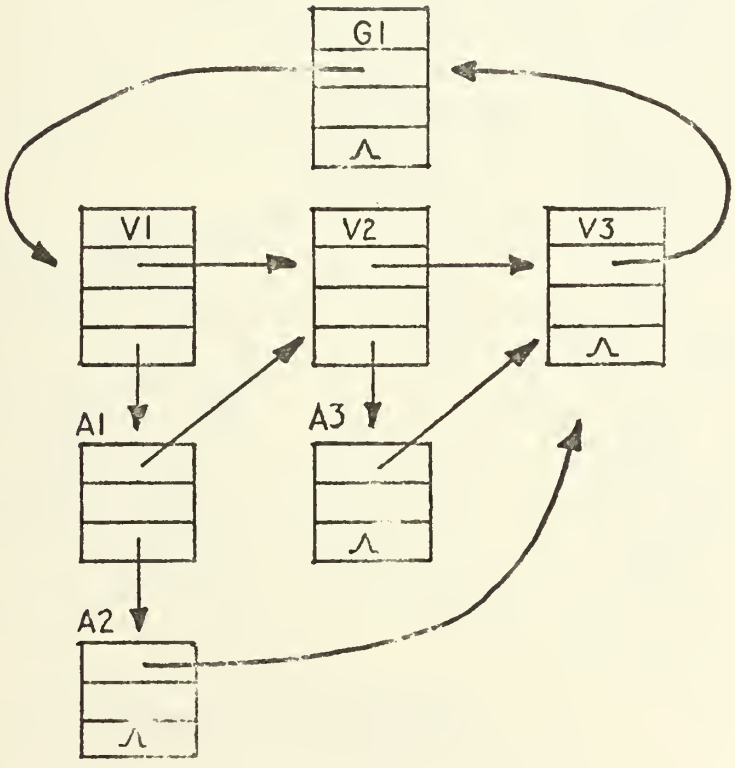
To facilitate processing, the NODE cells are put into a list linked by the NEXT pointer and another NODE cell called the graph header cell is added in which to store the name of the graph and to provide a starting point for processing the graph. Use of the NEXT pointer and addition of the graph header cell is shown in Figure 9.



(a) A graph G1



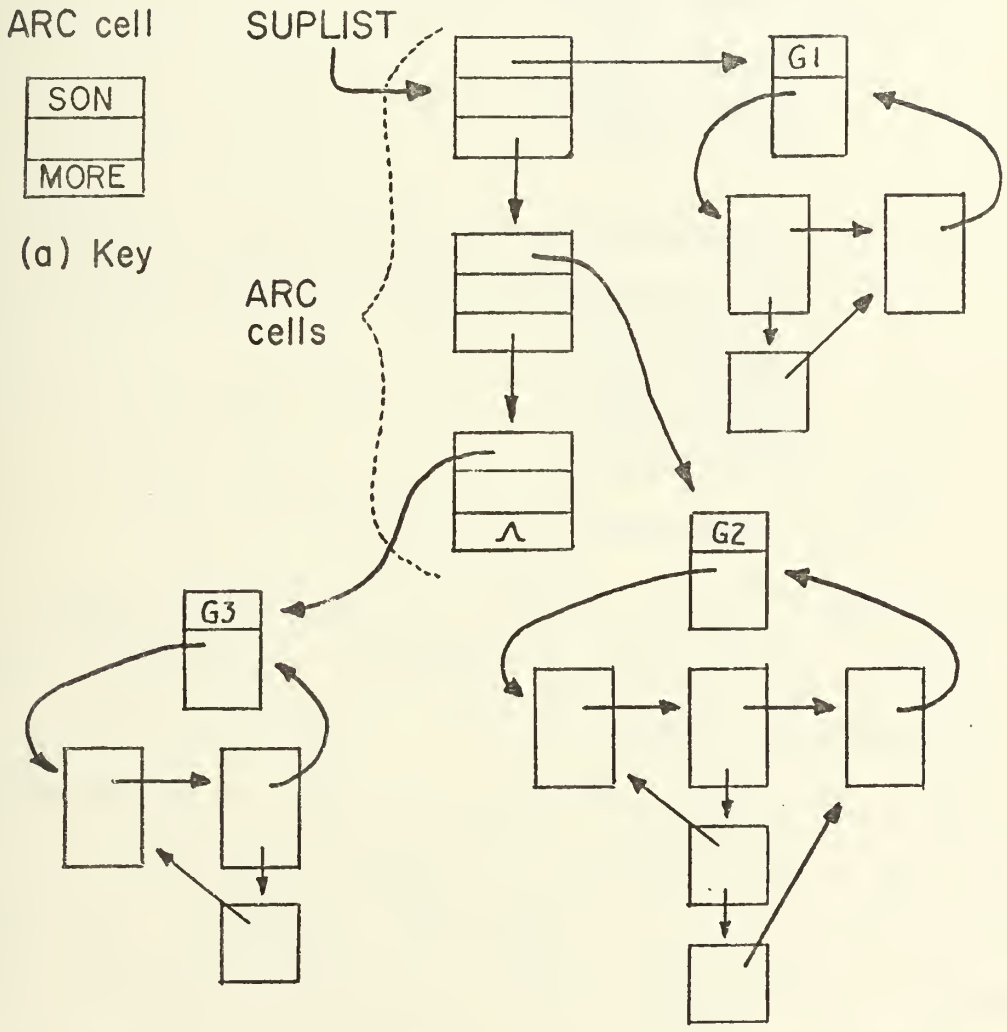
(b) Key



(c) List structure of graph G1

Figure 9. A graph G1 and its list structure

Because many graphs may be in core simultaneously, a list of ARC cells linked by the MORE pointer is maintained with the SON pointers identifying the graph header cells of the graphs in core. The system pointer variable SUPLIST (mnemonic for super list) points to the top of this list. Figure 10 shows the use of these ARC cells and the pointer SUPLIST.



(b) The list structure

Figure 10. A list structure showing the use of the system pointer variable SUPLIST

SUPLIST is set to the NULL pointer value if no graphs are in the system. Note that through the variable SUPLIST, the system and user routines can access all of the graphs active at any time.

To store the information necessary to indicate that a cell of a graph represents a super-node or super-arc, some additional fields are needed in the cells:

```
DECLARE 1 NODE BASED (PP),
          2 LABEL CHARACTER(4),
          .
          2 NEXT POINTER,
          2 SNODE POINTER,
          2 HINFO POINTER,
          .
          .
          2 DOWN POINTER,

        1 ARC BASED (P-ARC),
          2 SON POINTER,
          2 SARC POINTER,
          2 AINFO POINTER,
          2 PSARC POINTER,
          2 PAINFO POINTER,
          .
          2 MORE POINTER;
```

Consider Figure 11a and b. Graph G2 contains a NODE cell V11 which is a super-node representing a copy of graph G1. SNODE of the super-node cell points to G1's graph header cell, identifying the graph that it represents. An ARC cell labelled IH is added to the list of ARC cells pointed to by the DOWN pointer of G1's graph header cell. An ARC cell will appear on this list for each copy of the graph currently in the system. This ARC cell is called an interface header cell. Its SON field points to the super-node's NODE cell. Its MORE field points to more interface header cells of G1, NULL in this case. Its AINFO field points to a list of interface ARC cells. Figure 11d shows the linkage between a super-node cell V11, a graph header cell G1, and an interface header cell IH. The characters "(XXX)" in

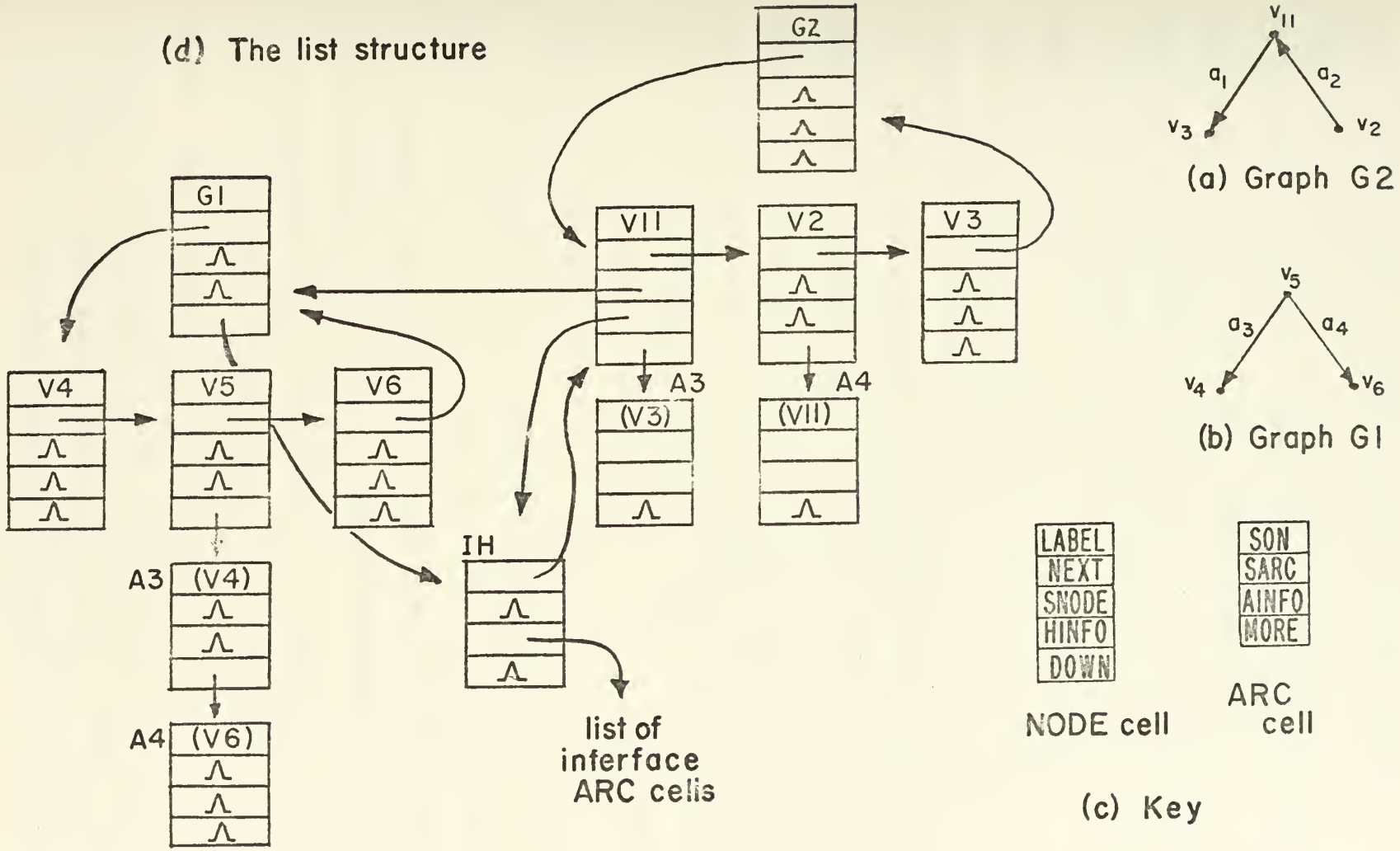


Figure II. A list structure showing super-node linkage

the drawing of a list indicate that the field is a pointer identifying the cell labelled XXX. All ARC cells have a label representing its address just above it.

The list of interface ARC cells form the "interface" between a super-node and a copy of the graph. This interface concept allows many copies of the graph to exist without using computer storage for a separate copy of the graph for each super-node. Also, processing time may be reduced significantly. Consider the analysis of a PERT network representing the construction of a fifty story office building. A graph F may be designed to represent the activities necessary for constructing one floor--one node in F for each activity. Then, another graph B which represents construction of the entire building is designed containing among other nodes, fifty super-nodes each representing the construction of a floor, graph F. The PERT routine would first determine critical path values for graph F and associate these values with each super-node representing graph F. Then, graph B is analyzed by the PERT routine.

If the graph F, on the other hand, were duplicated fifty times and merged into graph B, the analysis of graph F would occur fifty times. It is apparent that a similar saving in time and storage would be made by applying this concept to other problems (e.g., the analysis of an electronic circuit containing many "copies" of a simple amplifier).

The representation of the last piece of vital information--with which nodes in the graph G1 of Figure 11 are the super-arcs in G2 incident--is now unveiled. First, the ARC cells which start or end at v_{11} in G2 are super-arcs. Their SARC and AINFO fields were left blank in Figure 11. Also, the AINFO field of the interface header cell points

to a "list of interface ARC cells". This list and the empty fields of the super-arc cells define with which nodes the super-arcs are incident.

There is one ARC cell in this list for each super-arc starting or ending at v_{11}. An ARC cell in G2 is denoted as representing a super-arc by setting either its SARC or AINFO pointers (or both) to point to a unique cell on the appropriate interface list--depending on the starting and ending nodes in G2.

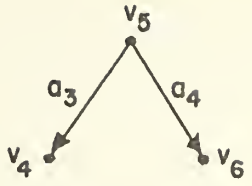
If the ending node is a super-node, then the SARC field of the super-arc cell is set to point to an interface ARC cell. The interface ARC cell is on the list of interface ARC cells identified by the HINFO field of the ending super-node cell. This interface ARC cell's SARC field points to the node in G1 at which the super-arc also ends.

If the starting node is a super-node then the AINFO field of the super-arc cell points to an interface ARC cell. This interface ARC cell is on the list of interface ARC cells identified by the HINFO field of the starting super-node cell in G2. The AINFO field of the interface ARC cell points to the super-arc's starting node in G1.

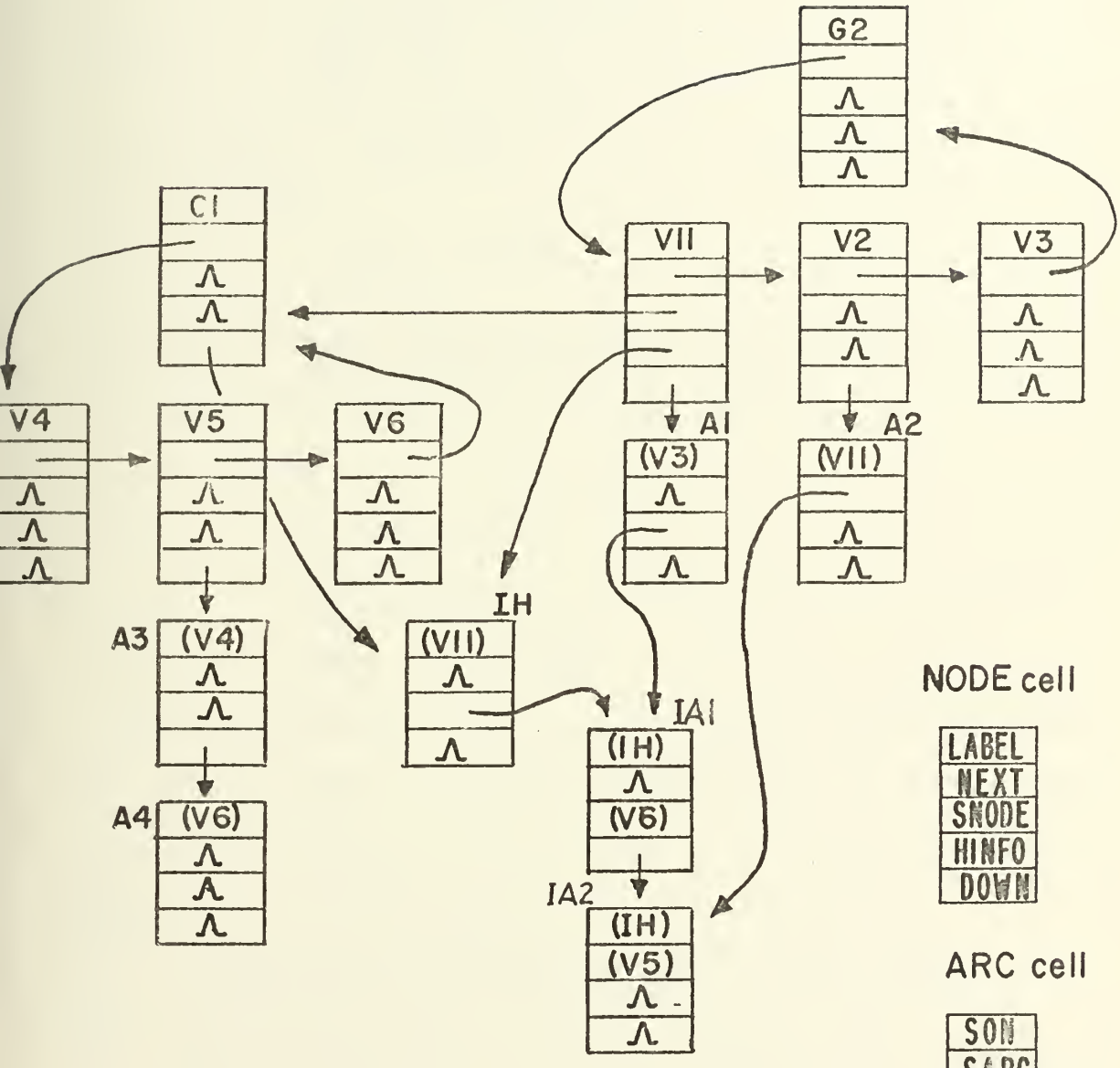
If both the starting and ending nodes of a super-arc are super-nodes, then pointers are set as described above. The SON field of an interface ARC cell points to its interface header cell. Figure 12 shows the linkage between the super-node V11, super-arcs A1 and A2, and the interface ARC cells IA1 and IA2. Cells labelled IAn are interface ARC cells, A1 and A2 are super-arc cells, and the cell labelled IH is an interface header cell.

An example showing a three graph system with a super-arc between two super-nodes, a super-arc starting at three nodes (each in a

(a) Graph G2



(b) Graph G1



(d) The list structure

NODE cell

LABEL
NEXT
SNODE
HINFO
DOWN

ARC cell

SON
SARC
AINFO
MORE

(c) Key

Figure 12. A list showing an interface ARC cell list

different graph), and showing multiple copies of a graph appears in Figure 13.

Graph G3 = (V_3, A_3, ∇_3),

where

$V_3 = \{v_{21}(G2), v_{12}(G1)\}$
$A_3 = \{a_4\}$
$\nabla_3(a_4) = ([v_{21}, v_{11}, v_1], [v_{12}, v_2])$

Graph G2 = (V_2, A_2, ∇_2),

where

$V_2 = \{v_4, v_{11}(G1)\}$
$A_2 = \{a_3\}$
$\nabla_2(a_3) = ([v_4], [v_{11}, v_3])$

Graph G1 = (V_1, A_1, ∇_1),

where

$V_1 = \{v_1, v_2, v_3\}$
$A_1 = \{a_1, a_2\}$
$\nabla_1(a_1) = ([v_1], [v_3])$
$\nabla_1(a_2) = ([v_1], [v_2])$.

Since super-arcs may start (or end) at several super-nodes, each in a different graph, the PAINFO (and PSARC) field is needed in which to store this information. The following set of rules determine the number and field values of interface arc and header cells:

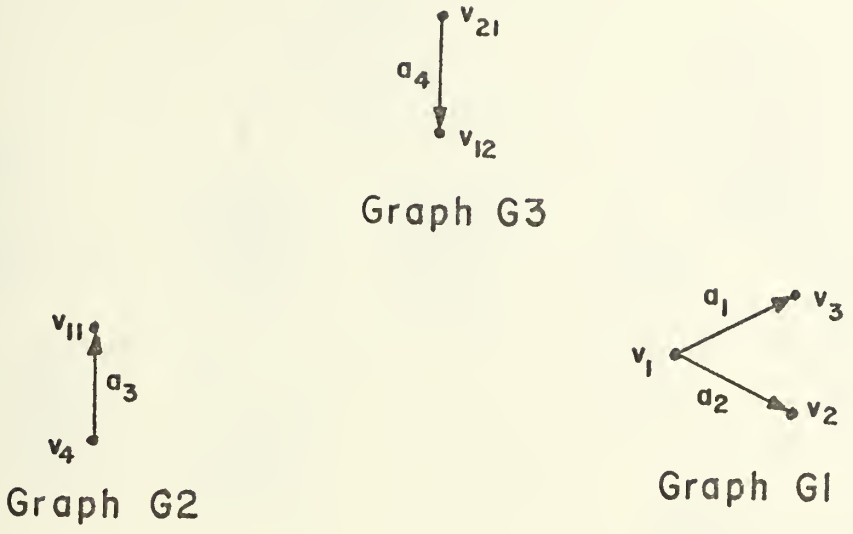
1) There exists an interface header cell IHij under Gi's graph header cell for each super-node SNij representing graph Gi.

2) IHij's SON points to a unique super-node SNij representing graph Gi.

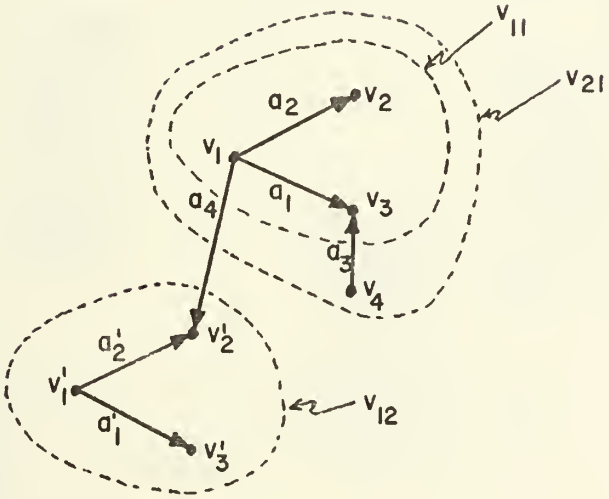
3) IHij's AINFO points to a list of interface ARC cells IAijk.

4) An interface ARC cell exists for each super-arc eventually starting or ending at SNij.

5) IAijk's SON points to IHij.



(a) Three graphs



(b) Graph G3 without super-nodes and super-arcs

Figure 13. Three graphs and the list structure

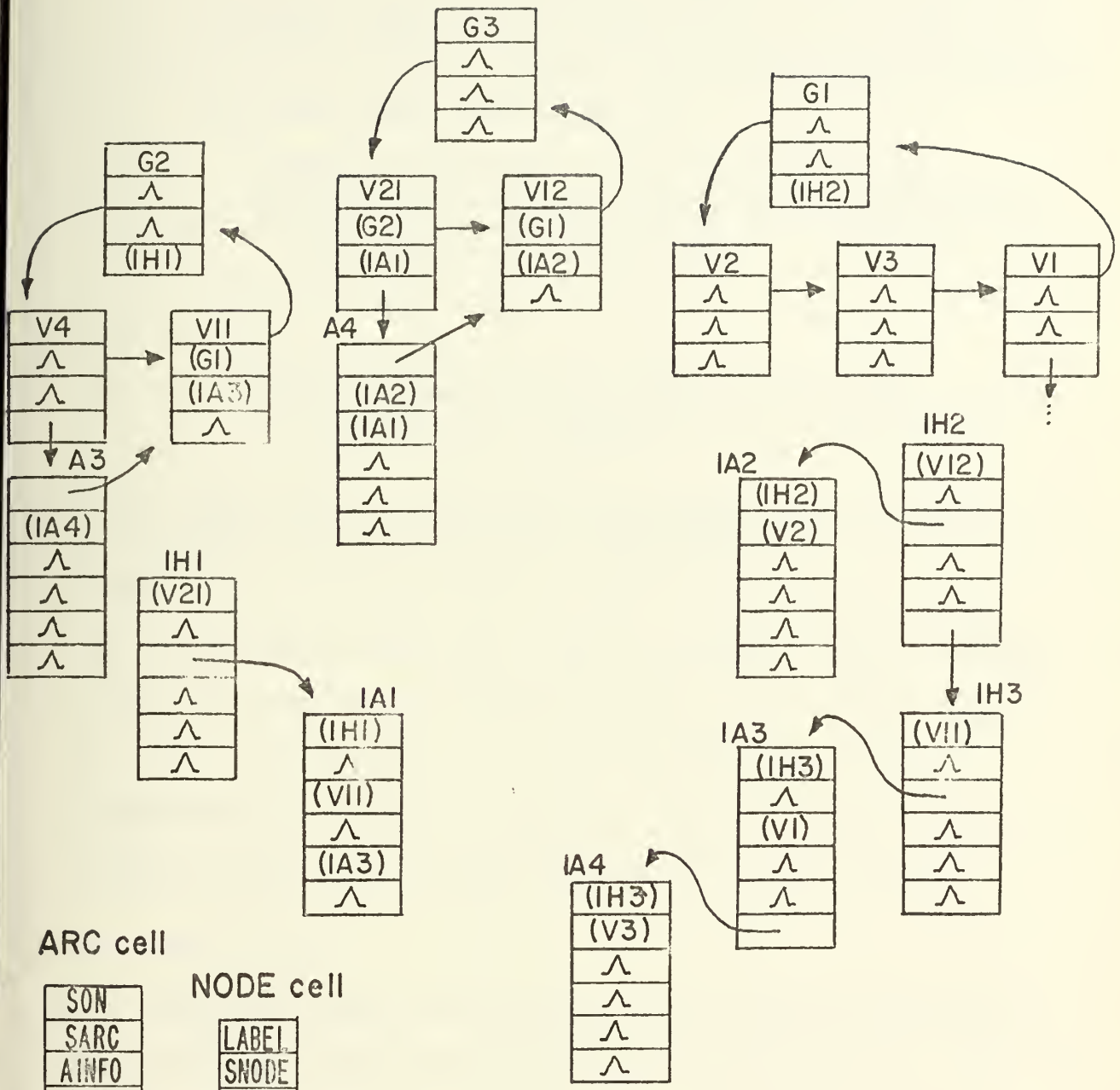


Figure 13. (Con't) Three graphs and the list structure

6) IAijk's SARC points to a node Nip in Gi if the kth super-arc in the system eventually incident with node Nip ends at node Nip.

7) IAijk's AINFO points to a node Niq in Gi if the kth super-arc in the system eventually incident with node Niq starts at node Niq.

8) IAijk's PSARC points to IAlmn if

a) IAijk's SARC points to a super-node SNlm in Gi.
b) The super-arc associated with IAijk also ends at SNlmn.
c) The kth super-arc incident with node Nip is also the nth super-arc incident with SNlm.

9) IAijk's PAINFO points to IAlmn if

a) IAijk's AINFO points to super-node SNlm in Gi.
b) The super-arc associated with IAijk also starts at SNlm.
c) The kth super-arc incident with Niq is also the nth super-arc incident with SNlm.

10) Otherwise, the fields in the interface ARC and interface header cells (except MORE) are set to the NULL value.

Figure 13d shows the list structure reflecting the use of interface ARC and interface header cells. Note, for example, that the starting node of A4 can be traced (through IA1) to V11, then (through IA3) to V1. Also, given a graph, say G1, one can immediately locate all super-nodes which represent the graph, V12 and V11, and from there, the super-arcs which start and end at nodes in the graph, A4 and A3.

D. USER ROUTINES

Because the system is not designed for a particular application, the task of applying it to an application is left to the user. The routines the user must provide are application-oriented and are divided into two classes. One class handles user-defined fields and cells. The other class consists of analysis routines. System routines are provided to assist the user in the programming and implementation of his routines. Also, system variables are declared "EXTERNAL". Therefore, their values may be accessed at any time.

User-defined fields and cells are set up by the system in the following manner. First, each NODE cell of the directed graphs has four fields which are available for his use. Each time a node is added to or deleted from a graph, a user routine named N_USER is called. A pointer to the cell and a variable set to one for add or two for delete is passed to this routine. This routine may then use a system routine to retrieve information from the display device on-line to set the fields reserved for his use. The following fields are set aside in the NODE cells: XCOR and YCOR are FLOAT BINARY(21), ON_OFF is BIT(2), and USER is a POINTER. The user may design and allocate storage for his own cells to hold more information related to a node and set USER to address his cells. This routine is called just before a node is deleted so that he may free the storage used for any of his cells.

Likewise, one pointer field in each ARC cell may be used: USERA. N_USER is called in a similar manner; however, the calling integer argument is set to three for add and to four for delete.

Thus, during analysis, the user may access these values stored in the cells, use them for computations and perhaps re-store new values to be displayed later.

After his graphs have been put into the system, he may press a function key. Another user routine is called: BUTTONS. The number of the function key pressed, 0 through 31, is passed to this routine. He may then perform an analysis of the graphs, display textual information using, if desired, a system routine DISPAGE, or cause a printed output of one or more graphs--again, optionally using another system routine P_ARCS.

The last routine named ANALYZE must be provided. It is invoked by positioning the light pen on the displayed option: ANALYZE. Again, he may initiate any of the actions above (including calling BUTTONS with a dummy function key number!).

The user's manual in Appendix A contains a detailed discussion of the system. The program listing of an initial implementation on the IBM 360/67 appears in the Computer Program Section.

E. A SAMPLE ALGORITHM

To demonstrate how the list structure can be traversed, algorithm P is shown which will construct the path reachable set $P^n(v)$, given two arguments: N, the length of path progressions considered and PN, a pointer identifying the NODE cell representing v.

First, an algorithm is shown which checks an array of pointers to see if the last one is a duplicate of any of the pointers above it. This algorithm, named CHECK, will be called in algorithm P to determine whether the arc progressions generated in P are also path progressions. In the algorithm CHECK, POINTERS is an input one-dimension array of pointers. LAST is the index of the last element in the POINTERS array. POINTERS(1) is the first element. RESULT will be set to one if the last element in POINTERS is not in the array twice; it will be set to two if the last element is duplicated.

```
ALGORITHM CHECK(POINTERS,LAST,RESULT)

1.  RESULT <—— 1
2.  IF LAST is less than 2 THEN RETURN
3.  I <—— 1
4.  IF POINTERS(I) equals POINTERS(LAST)
                then GO TO STEP 8.
5.  I <—— I + 1
6.  IF I equals LAST THEN RETURN
7.  GO TO STEP 4.
8.  RESULT <—— 2
9.  RETURN
```

Algorithm P constructs all possible arc progressions of the length desired. WSEQ is a one-dimension array of pointers identifying the arc progression. IW is an integer indicating the last element in WSEQ used. Each time an arc is added to WSEQ, algorithm CHECK is called to determine whether the arc is already on the list. If so, the arc progression is not a path progression; thus, the algorithm explores another arc progression. If a progression with no repeating arcs contains N arcs then the ending node of the last arc in WSEQ (if not equal to PN) is an element of the path reachable set desired; the pointer to its NODE cell is added to the output array of pointers RSET. IR is similar to IW, indicating the last element in RSET.

The pointer array SAVEP and the integer array SAVEN stores pointers to ARC cells and the current value of IW respectively. These arrays are used to construct another arc progression. The index of the last element of both of these arrays is stored in the integer variable IS. ARC is a pointer variable identifying an ARC cell. ND is a pointer variable identifying a NODE cell.

```
ALGORITHM P(PN,N,RSET,IR)

    I1.  IW <--- 0
    I2.  IS <--- 0
    I3.  IR <--- 0
    I4.  ARC <--- PN's DOWN
    I5.  IF ARC = NULL THEN RETURN

    S1.  IF ARC's MORE = NULL THEN GO TO STEP W1.
    S2.  IS <--- IS + 1
    S3.  SAVEP(IS) <--- ARC's MORE
    S4.  SAVEN(IS) <--- IW

    W1.  ND <--- ARC's SON
    W2.  IW <--- IW + 1
    W3.  WSEQ(IW) <--- ARC
```

```
T1.  CALL CHECK(WSEQ,IW,RESULT)
T2.  IF RESULT = 2 THEN GO TO STEP P1.
T3.  IF IW = N THEN GO TO STEP A1.
T4.  IF ND's DOWN = NULL THEN GO TO STEP P1.
T5.  ARC <—— ND's DOWN
T6.  GO TO STEP S1.

P1.  IF IS = 0 THEN RETURN
P2.  ARC <—— SAVEP(IS)
P3.  IW <—— SAVEN(IS)
P4.  IS <—— IS - 1
P5.  GO TO STEP S1.

A1.  IF WSEQ(IW)'s SON = PN THEN GO TO STEP P1.
A2.  IR <—— IR + 1
A3.  RSET(IR) <—— WSEQ(IW)'s SON
A4.  CALL CHECK(RSET,IR,RESULT)
A5.  IF RESULT = 1 THEN GO TO STEP P1.
A6.  IR <—— IR -1
A7.  GO TO STEP P1.
```

Steps I1 through I5 initialize the system; S1 through S4 save pointers necessary to construct the next arc progression to be investigated; and W1 through W3 add an arc to the arc progression in WSEQ. Steps T1 through T6 check the progression to insure that no arc is duplicated and if N arcs are in the progression, then control is transferred to step A1. P1 through P5 resets ARC and IW so that another progression is checked. If no more are to be checked, the algorithm terminates returning RSET and IR. A1 through A7 check to insure $v_1 \neq v_{n+1}$ and if not, v_{n+1} is added to RSET permanently if not already in RSET. The next progression is then explored.

This algorithm can be modified to return the cumulative (path) reachable set or the (path) reachable set of a sequence of nodes.

IV. CONCLUSION

The system designed provides an input/output interface between the directed graph problem and the computer. Since pictures of graphs often are heuristically useful in obtaining solutions to some problems, a graphic display unit is used on which to present a drawing of the graphs. Also, through the unit's light pen and function keyboard, the user may interact with the graphs--thus providing an on-line graph design capability. Upon specified interrupt actions, user-written analysis routines are invoked which have access to the list structure built and maintained by the system to represent the directed graphs. System routines which may be invoked by user-written procedures are provided to aid in the design and implementation of the system for a specific application.

Because the system is not designed for a particular application, a non-trivial programming task is left to the user. Also, the fact that the storage for the list structure is dynamically allocated means that the user-written programs should be written in either PL/I or assembler. List processing, in itself, usually is not undertaken by the novice programmer. Therefore, the field of probable users is somewhat limited.

The PL/I language of the IBM 360/67 restricts the total amount of storage dynamically allocated to based structures to 16^6 bytes. Therefore, when this storage is filled, no more nodes or arcs may be allocated. Hence, a sub-system handling the temporary storage of graphs on a mass storage medium is a necessary addition to the system to allow the analyses of large graphs.

Also, the number of nodes and arcs displayed is limited by the size of the buffer of the IBM 2250 Display Unit. A 4K, 8K, 16K, or 32K byte buffer may be installed in the 2250. A 4K buffer allows approximately twenty nodes and arcs to be displayed at any one time.

At the time of publication of this paper, many IBM 360/67 systems are replacing the IBM Operating System with IBM's TSS which does not support PL/I or the 2250. However, IBM has announced that both will be supported eventually. Random access is not to be supported under TSS. This may severely restrict the use of a mass storage medium for temporary storage of directed graphs.

Other graph theoretic systems have been designed. An extension to ALGOL was designed by S. Crespi-Reghizzi and R. Morpurgo [2] to allow specification and manipulation of graphs in the source language. It is not interactive, does not allow the representation of many nodes by one node and does not provide a display capability. It does use dynamic storage allocation and allows directed and undirected graphs. Real numeric fields may be associated with nodes and arcs.

Michael S. Wolfberg [3] designed and implemented an interactive graph theory system on the IBM 7040 with a DEC-338 graphics terminal. The abstract from his paper indicates that his system provides, in addition to the graph manipulation options of this system, the capability of entering analysis routines from the terminal. The abstract did not indicate whether dynamic storage allocation or the super-node concept is used.

A system used in manipulating trees was designed and implemented on the IBM 360/67 with a 2250 Display Unit by Claude Holifield [4]. His thesis was not available for comparison at the time of publication of this paper.

APPENDIX A

USERS' MANUAL

Interactive Graph Reduction and Analysis Program

USERS' MANUAL

by

Lieutenant (junior grade) James W. Thomas, USN

TABLE OF CONTENTS

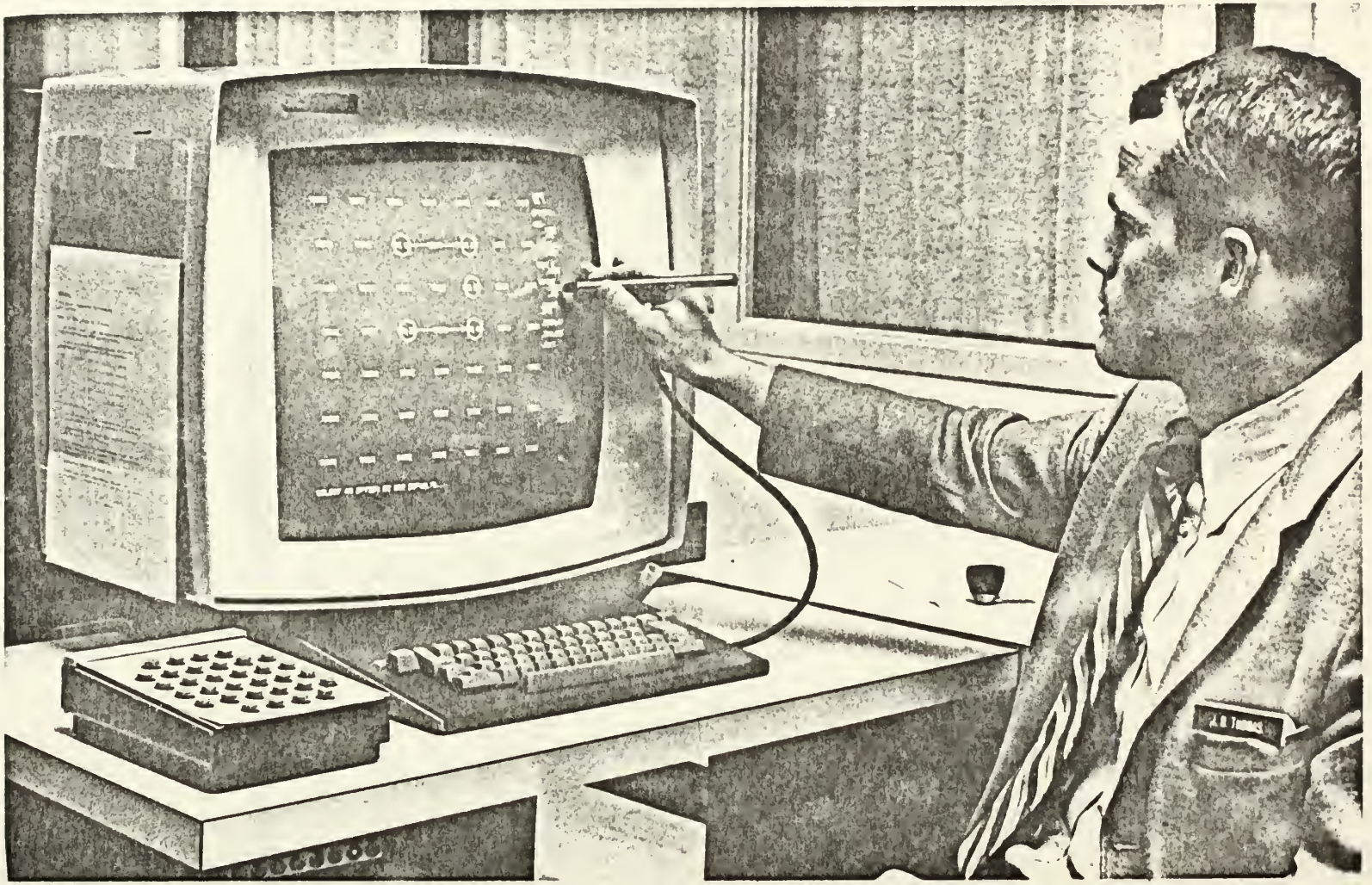


Figure 1

The IBM 2250 Display Unit. Using the function keyboard on the left, the alphanumeric keyboard below the screen, and the light pen, the user may interact directly with the system.

I. INTRODUCTION

The Interactive Graph Reduction and Analysis Program provides the user with a software package which aids in the input specification of directed graphs to computer core storage, allows on-line modification of the graphs, and submits a PL/I list structure representing the directed graphs to user-written analysis routines.

The package is written in version 4.3 of IBM's PL/I (F) programming language and runs under release eighteen of the IBM 360/67 Operating System. It uses the IBM Graphic Subroutine Package (Program Number 360S-LM-537) to support the IBM 2250-1 Display Unit and executes in less than 200K of core storage.

The system allows on-line design of directed graphs. The user may input a directed graph through the display unit, then request his analysis routines to be invoked. Output from the analysis may be displayed and used by him to determine changes he may want to make in the graph. This interactive process may continue indefinitely.

The user should be familiar with the following IBM publications:

IBM System/360 Component Description: IBM 2250 Display Unit Model 1. Form A27-2701

IBM System/360 Operating System Graphic Subroutine Package (GSP) for FORTRAN IV, COBOL, AND PL/I. Form C27-6932

IBM System/360 PL/I (F) Language. Form C28-8201

IBM System/360 Operating System PL/I (F) Programmer's Guide. Form C28-6594

IBM System/360 Operating System Linkage Editor and Loader. Form C28-6538

IBM's Job Control Language cards needed to execute the system are shown in the Computer Program and Computer Output sections of this paper.

II. DISPLAY FRAMES

At the IBM 2250 Display Unit, the user works with four system-generated pictures or display frames. Each frame handles part of the total system task. The frames are listed below with a brief description of each. They are discussed in detail in Chapter III.

A. INPUT MODE SELECTION FRAME

This is the first frame displayed by the system. It allows the user to choose the initial input mode: cards, magnetic device or the display unit. The user utilizes the light pen to choose the mode of input desired.

B. GRAPH MANIPULATION FRAME

This is the frame used to display and alter the directed graphs. Graphs may be input through this frame. User-written analysis routines are called from this frame.

C. GRAPH SELECTION FRAME

Often many graphs may be in the system at any one time. However, only one may be displayed in the Graph Manipulation Frame. If the user wants to display a graph, this frame presents the names of all graphs in the system and allows the user to detect with the light pen the graph he wants displayed on the Graph Manipulation Frame.

D. USER INPUT/OUTPUT FRAME

This frame is utilized to display a "page" of text generated by a user-written routine and return light pen, function keyboard and alpha-numeric keyboard information to the calling routine.

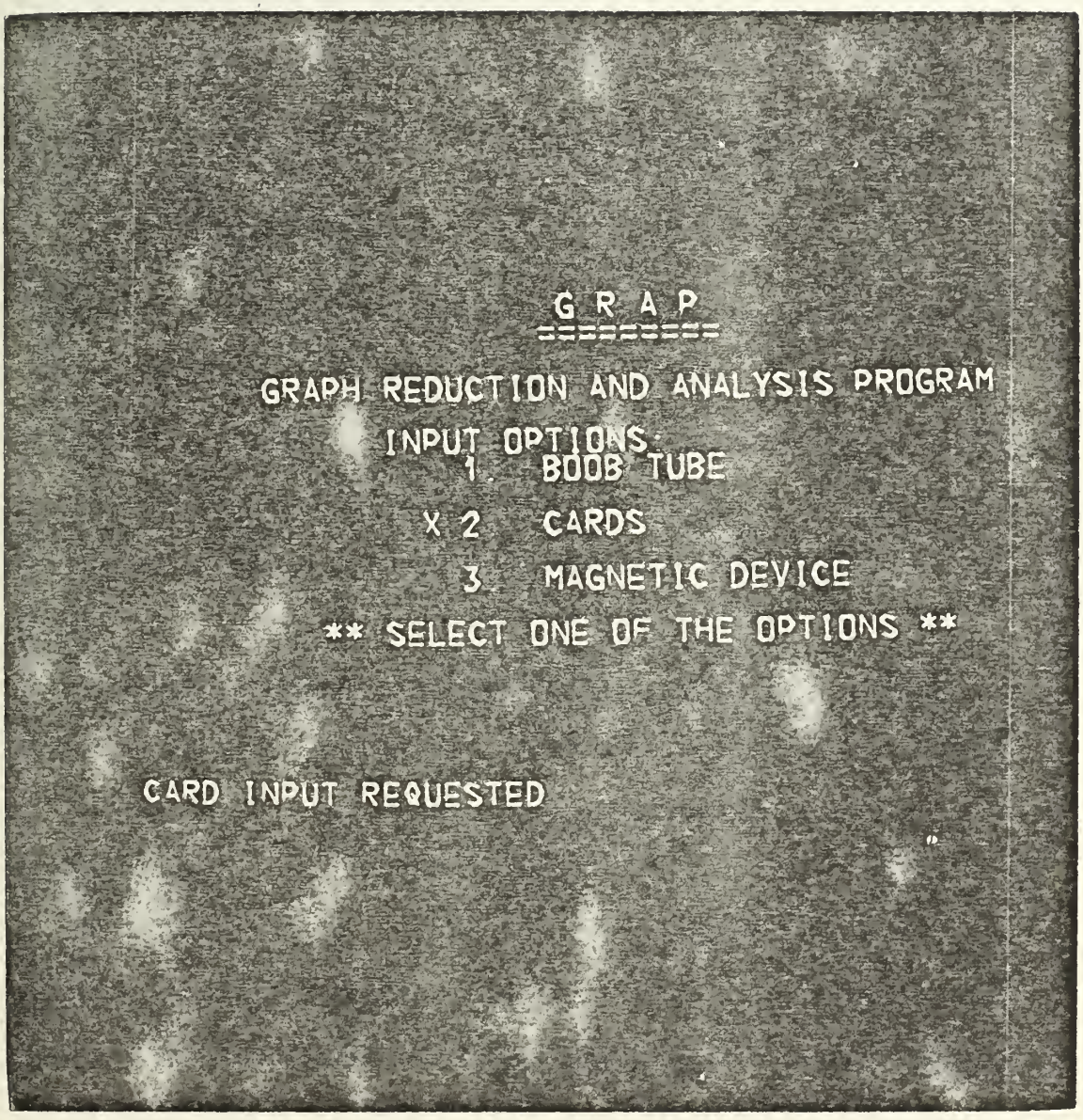


Figure 2

The Input Mode Selection Frame. Positioning the light pen on option 1, 2 or 3 relays the user's request to the system. Above, option 2, CARDS, had been selected.

III. DISCUSSION OF THE DISPLAY FRAMES

A. INPUT MODE SELECTION FRAME

This frame displays the three possible sources of input to the system. The user selects a source by pointing the light pen at

either CARDS
MAGNETIC DEVICE
or BOOB TUBE.

Depressing the foot pedal activates the light pen detection facility and the selection is transmitted to the system. If the light pen is not directed at one of the three options when the pedal is pressed, the user is notified by the replacement of the row of asterisks at the bottom of the display with the message, "INVALID OPTION...TRY AGAIN."

If CARDS is selected, the system reads a "packet" of data cards submitted as file SYSIN of the job step which invoked the execution of the system. The data card format is described in a later section of this manual.

If MAGNETIC DEVICE is selected, the system reads a similar data packet created by a previous run. This file is named MAGINPT and is a sequential card image file on any magnetic mass storage device.

If BOOB TUBE is selected, the system displays the Graph Manipulation Frame for on-line input of graphs. The user may input a graph through the CARDS option, then select this mode to change the graph.

B. GRAPH MANIPULATION FRAME

This frame is divided into three regions. One region, along the right-hand margin of the tube, is used to display a list of options the

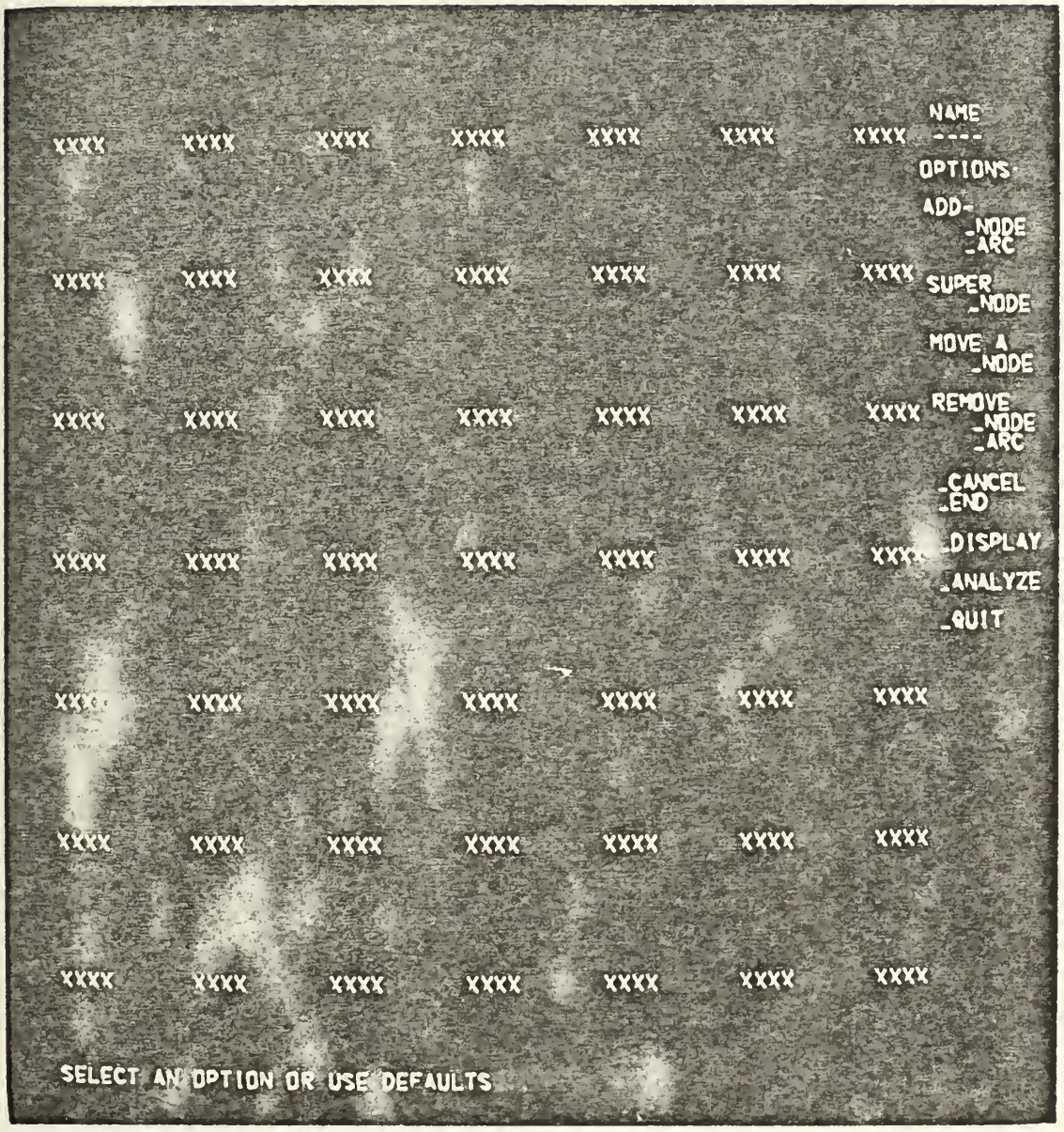


Figure 3

The Graph Manipulation Frame showing 49 possible node positions indicated by four X's. The frame is ready for input of a graph to the system. Note the four underscores under "NAME". These are replaced by the graph name. The options appear to the right of the node positions. In a recent change to the program, "SAVE" replaced "END" in the option list.

user may select with the light pen. These options or commands direct the system in manipulating the graphs. Another region is used to display messages to the user. The third and largest region is used to display a graph consisting of octagons or nodes and arrows or arcs between octagons. Inside each node is a four-character label assigned by the user or a default label "NNxx" where xx is a unique decimal integer. The first two characters are restricted to alphabetic characters.

The options allow the user to add and delete nodes and arcs, move a node from one of the forty nine possible positions to another, combine some of the nodes displayed into a graph replacing them by a super-node and to add nodes to graphs represented by super-nodes. The user may add and delete super-nodes and store a displayed graph, clearing it from the frame, so that a new graph may be entered. He may cause his analysis routines to be called at any time which will have access to ssystem pointer variables. A more detailed discussion of the options and their effects follows.

As mentioned above, the Graph Manipulation Frame is divided into three regions: The GRAPH DISPLAY AREA consists of forty nine possible locations where a node may be displayed. Each location is denoted by the display of four X's. If a node occupies a location, then the X's are replaced by the node label and an octagon is drawn around the label. Arcs represented by arrows may be displayed between any two nodes.

The USER MESSAGE AREA is used to display messages to the user. Upon selecting an option, instructions are displayed in this area telling the user which actions are expected and what errors have been made.

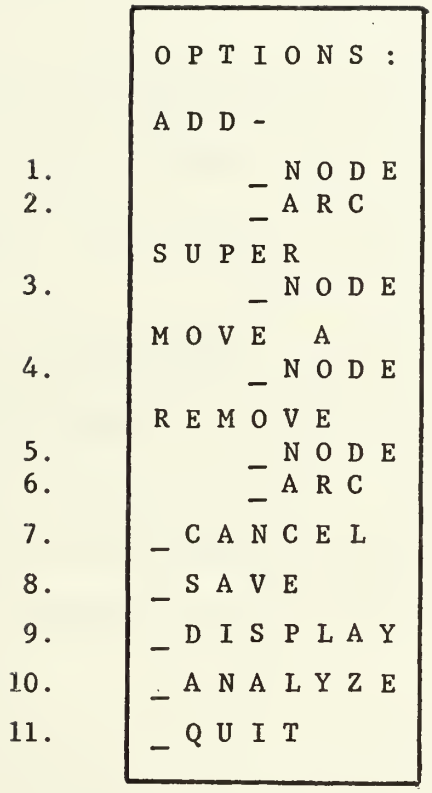


Figure 4

The option list of the Graph Manipulation Frame. The options provide the capability of altering the graph displayed in the frame. The options are numbered here to provide a reference for the description of the options in Chapter III. Note that only lines numbered above may be detected by the light pen.

The OPTION AREA is used to display options that the user may select with the light pen. Any of the textual lines preceded by an underscore in the option list may be detected by the light pen. Those lines are numbered in Figure 4. The user may make the following option selections:

1. ADD A NODE

Selection of this option allows the addition of a node (simple or super-node) to the graph currently displayed. If the screen is clear and the graph name (appearing just below the text "NAME" and just above "OPTIONS:") is "_ _ _ _", then the system will insert a cursor in this graph name area under the first underscore and request in the user message area that the user enter a name for the graph. After the name has been entered, pressing any function key or generating a light pen detect on the name will instruct the system to go on to the next step. The next step is the first step executed if the graph is already named.

Now, the system will instruct the user to point the light pen at a vacant node position denoted by four X's. If anything else is detected (except _CANCEL), the system displays an error message and requests the user to try again. When a valid available node position has been detected, a cursor is inserted under the first X and a new node label is requested. The characters typed by the user on the alphanumeric keyboard will replace the four X's. If the X's are not replaced, the system will provide a default name consisting of two N's and a two digit number. Pressing any function key or generating a light pen detect on the label will instruct the system to continue. An octagon will then be drawn around the label and the system will

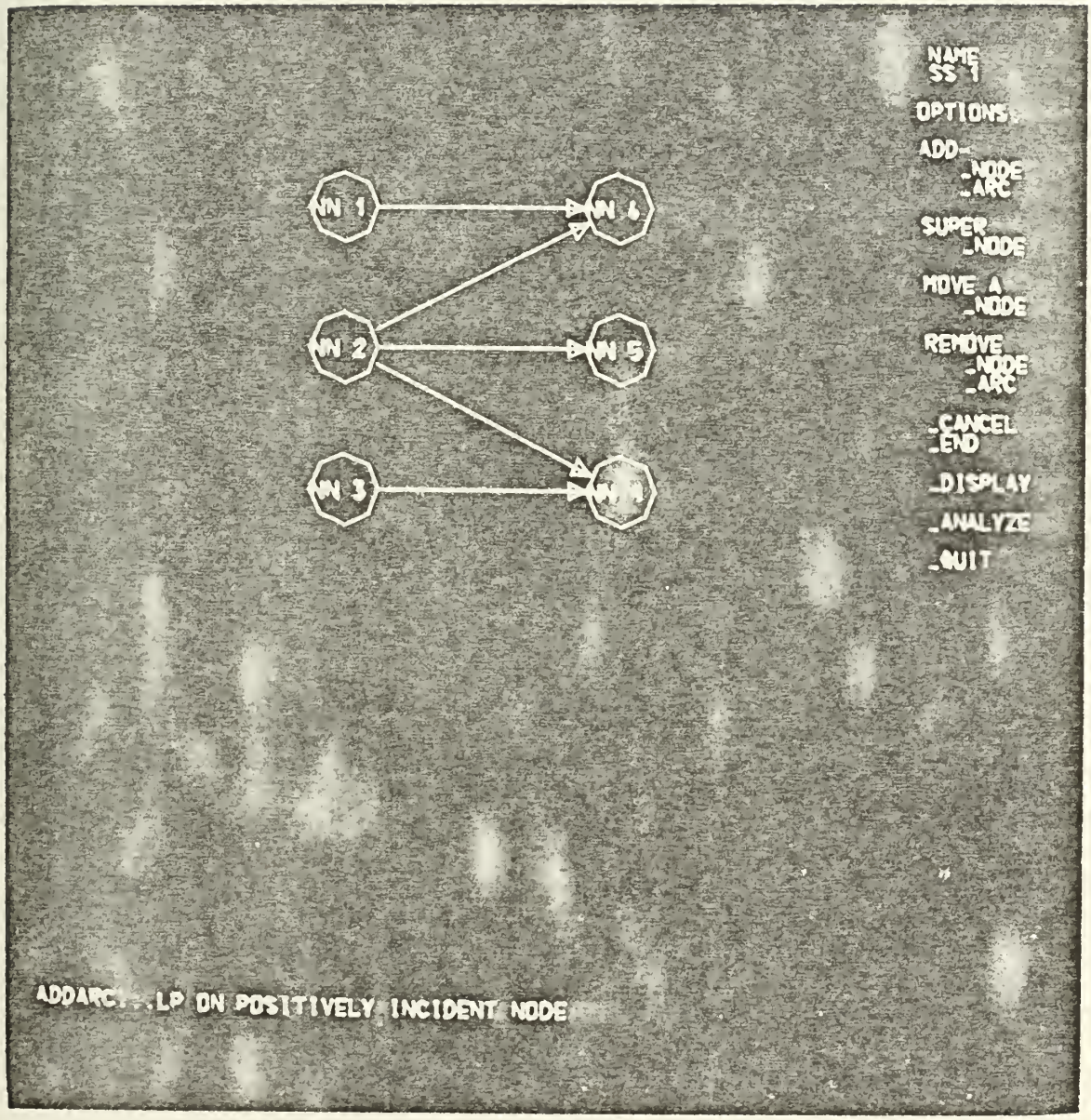


Figure 5

The Graph Manipulation Frame displaying graph SS 1. ADD AN ARC option had just been selected. Note the message to the user requesting the light pen (LP) to be positioned on the starting node of the arc to be added.

return to the ready state with the message displayed: SELECT AN OPTION OR USE DEFAULTS. The default options are described later.

If the node is to be a super-node, after entering the node label, a light pen detect on the SUPER-NODE option will cause the system to display the Graph Selection Frame, asking for a light pen detect on the name of the graph to be represented by the super-node. After this selection is made, the system will return to the Graph Manipulation Frame with the super-node added.

2. ADD AN ARC

Selection of this option allows the addition of an arc (simple or super-arc) to the graph currently displayed. If no graph is displayed a message requesting the user to select ADD A NODE is displayed and the system returns to the ready state. If only one node is displayed when this option is selected, a message alerts the user to select ADD A NODE before an arc may be added; loops are not allowed.

If at least two nodes are on the screen, the user is requested to select (with the light pen) the starting node and the ending node for the arc. An arrow will be displayed between the two nodes, pointing from the starting node to the ending node. If either or both nodes are super-nodes, the graphs representing these nodes will be displayed; the user will be requested to select a starting (or ending) node for the super-arc in these graphs and so forth until the user has eventually specified a simple node as the starting (or ending) node.

3. SUPER-NODE

Selection of this option when the system is in the ready state will cause the graph currently displayed to be saved and the screen cleared ready for the entry of another graph. If another data packet

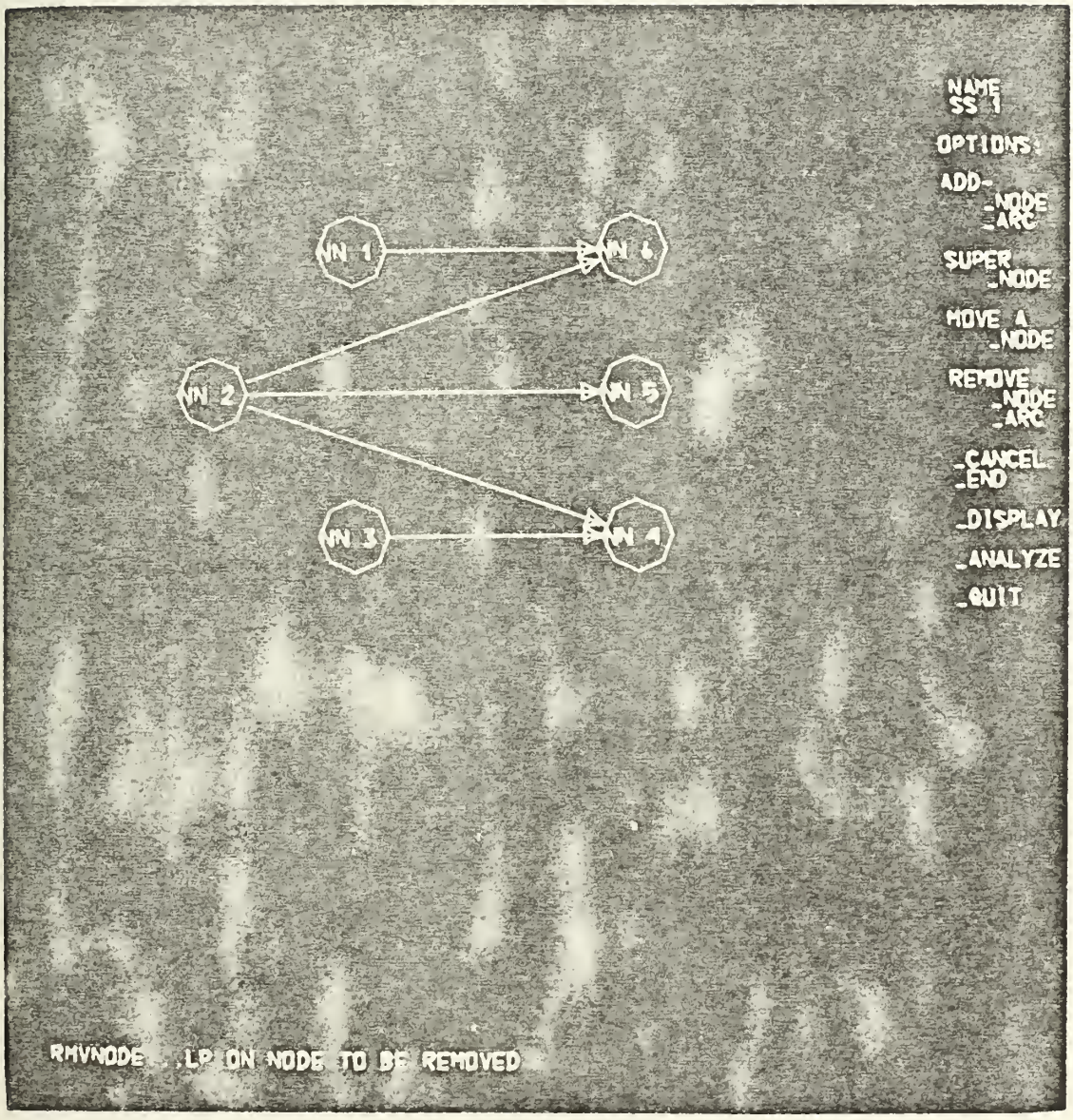


Figure 6

The Graph Manipulation Frame showing the result from moving NN 2 from its position in Figure 5 to its position here. The arcs were moved by the system. Note the REMOVE A NODE option had just been selected. The message to the user is requesting the light pen to be positioned on the node to be removed.

is waiting to be read, the system will return to the Input Mode Selection Frame. The user may select the desired mode.

4. MOVE A NODE

This option is included for aesthetic reasons only. Selection of this option allows the user to change the physical (displayed) location of a node on the screen. The list structure representing the graph is not altered. Upon selection of this option, the user will be asked to select the node he wishes to move, then to select a vacant node position. The node and all incident arcs will be moved to the desired position.

5. REMOVE A NODE

Selection of this option allows the removal of a node from the screen and the list structure. The user will be asked to indicate the node he wishes to be removed from the display and the list structure. Upon detecting a node (not a vacant position), the node and all arcs starting or ending at the selected node will be removed. The interface ARC cell list(s) will be updated if the node to be removed or any node incident with arcs removed are super-nodes.

6. REMOVE AN ARC

Selecting this option will cause the arc subsequently detected to be removed and the list structure to be updated. A detect on anything (except _CANCEL) will be flagged as an error and the user will be requested to try again.

7. CANCEL

This option may be used to return the system to the ready state cancelling the last option selected. For example, if the user detected the option ADD AN ARC and then realized that he had not added a node

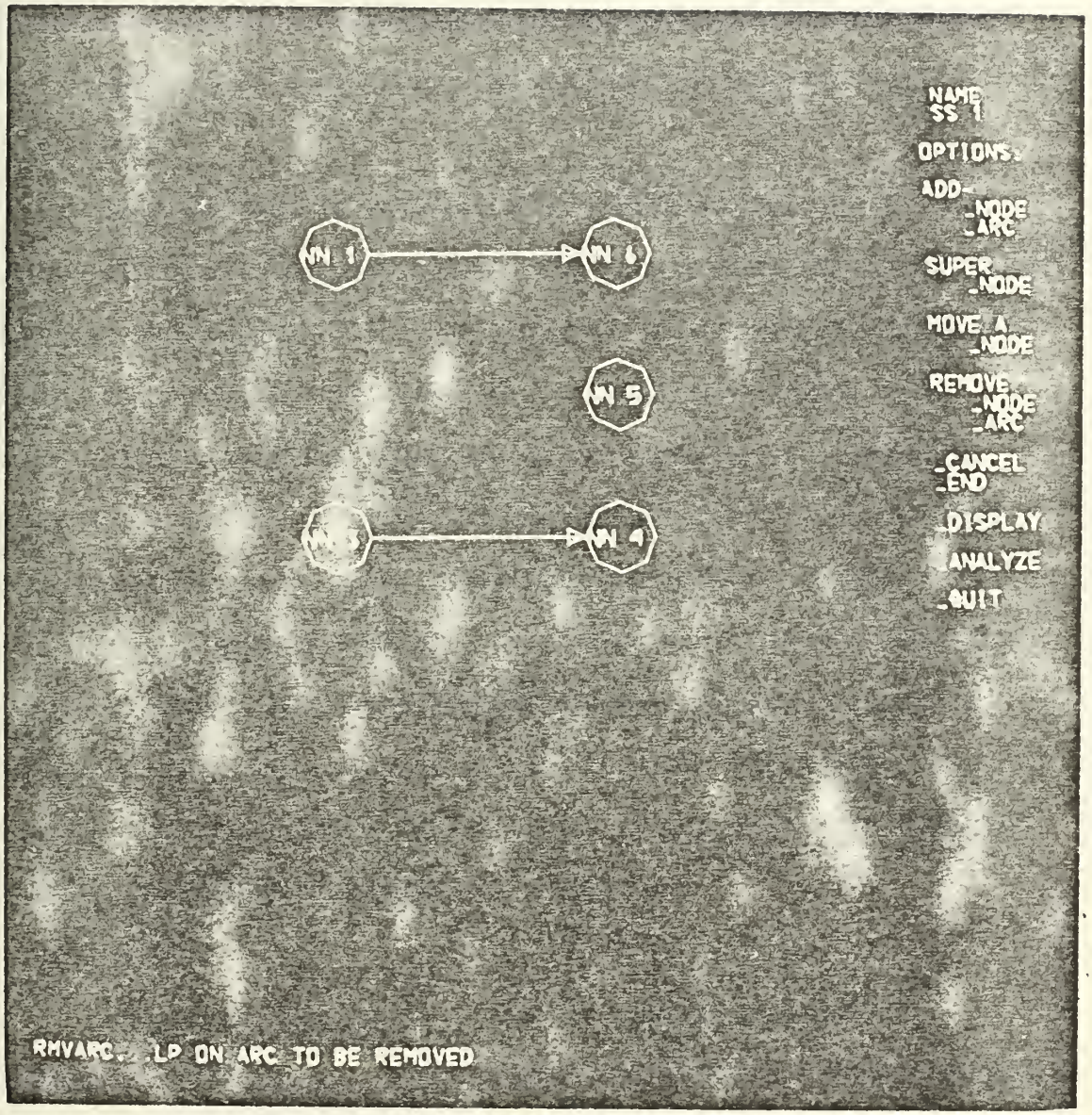


Figure 7

The Graph Manipulation Frame showing the result from removing NN 2 from its position in Figure 6. The arcs incident with NN 2 were removed by the system with the node. The user had just selected the REMOVE AN ARC option. Positioning the light pen on an arc would have removed it from the screen and the graph.

with which the arc was to be incident; he would then detect this option. The system would return to the ready state, cancelling the ADD AN ARC command. He would then add the missing node then add the arc. CANCEL may be used during the execution of any option.

8. SAVE

When the user is finished with a 2250 session, he may save some or all of the directed graphs that he presently has in the system. Selecting this option will cause the Graph Selection Frame to be displayed along with the word "ALL" at the right-hand margin of the screen. Positioning the light pen on the word ALL will cause all graphs to be saved. The user may selectively choose with the light pen those graphs to be saved if he does not want all of them saved.

When a graph is "SAVED", the system writes in a sequential card image output file named SAVEFLE one data packet representing the graph. This file may then be used as input file MAGINPT in conjunction with the MAGNETIC DEVICE option of the Input Mode Selection Frame to initialize the system for a later run.

9. DISPLAY

Selecting this option allows the user to re-display a previously saved graph. If after selecting DISPLAY, the user detects SUPER-NODE, the Graph Selection Frame will be displayed. The user may select a graph which is to be displayed in the Graph Manipulation Frame. He may then alter that graph.

If, after selecting this option, the user positions the light pen on a node or arc, additional information will be displayed about that node or arc. Detecting _CANCEL will return the system to the ready state.

If after selecting this option, the user positions the light pen on the name of the graph just above the text "OPTIONS:", information about the super-nodes representing the graph is displayed. Again, a light pen detect on _CANCEL will return the system to the ready state.

10. ANALYZE

Selecting this option will cause the system to call the user procedure named ANALYZE. See the chapter "User Routines" for further discussion.

11. QUIT

This option terminates execution of the system. Before detecting this option the user should save those graphs that he may want to use to initialize the system at the beginning of the next run. See the SAVE option of the Graph Manipulation Frame.

C. GRAPH SELECTION FRAME

When the user desires to display on the Graph Manipulation Frame a directed graph which is in the system but not currently displayed, or desires to add a super-node to a graph, he will be presented with the Graph Selection Frame. This frame consists of a list of graphs in the system; any one of which he may select with the light pen. If the last item in the list is "*MORE", then the number of graphs in the system is such that not all graph names may be displayed at once. If the desired graph is not on the list displayed, "*MORE" may be detected. This will cause another group of graph names to be displayed. When the last graph name has been displayed, "*END*" will be displayed instead of "*MORE." If the "*END*" is detected by the light pen, the first list of graph names will be re-displayed. A detect on the _CANCEL

option will cause the command requiring use of the Graph Selection Frame to be cancelled; the frame from which it was called will reappear.

D. USER INPUT/OUTPUT FRAME

The user may use this frame to display textual information and accept input from the alphanumeric keyboard, light pen or function keyboard. A call to the system routine which will display this frame follows:

```
DECLARE LINE(52) CHARACTER(74),
        (CODE,POS) FIXED BINARY(31),
        DISPAGE ENTRY((52)CHAR(74),FIXED BIN(31),
                      FIXED BIN(31));

CALL DISPAGE(LINE,CODE,POS);
```

The characters in LINE(1) are displayed along the top of the screen, LINE(2) immediately below LINE(1) and so forth. The other calling arguments are explained in the chapter "System Routines and External Variables."

IV. SYSTEM ROUTINES AND EXTERNAL VARIABLES

This chapter describes the system procedures and external variables which may be of value to the user in writing his routines.

A. EXTERNAL VARIABLES

1. SUPLIST (pointer) points to a list of ARC cells. Each ARC cell's SON field points to the graph header cell of a graph in the system. The cells in this list are linked by the MORE field of the ARC cells. This variable and the associated list may be used to access all the graphs in the system.

2. TOP (pointer) points to the graph header cell of the graph currently displayed in the Graph Manipulation Frame. If a graph is not displayed, this pointer is set to the PL/I NULL pointer value.

3. MES (character(40)) is a forty-character string whose contents are displayed in the user message area of the Graph Manipulation Frame by calling the system routine UPMES2.

B. SYSTEM ROUTINES

1. ALLOC allocates either an arc or node cell; initializing the pointer fields to the NULL value, the character string LABEL (if a node cell) to four blanks, and numeric fields to zero. The calling sequence is as follows:

```
DECLARE TYPE FIXED BINARY(15),
        POINT POINTER,
        ALLOC ENTRY(FIXED BIN(15),PTR);

CALL ALLOC(TYPE,POINT);
```

If TYPE equals 1, a NODE cell is allocated and initialized. If TYPE equals 2, an ARC cell is allocated and initialized. In either case,

POINT will identify the cell allocated upon return to the calling procedure.

2. P_ARCS prints on file SYSPRINT all nodes and arcs of a graph specified by its calling arguments. Starting and ending simple nodes and their respective graphs are also printed for super-arcs in the graph. User-defined variables associated with each node and arc are also printed at the user's disgression. The calling sequence is as follows:

```
DECLARE POINT POINTER,
        MESG CHARACTER(n),
        P_ARCS ENTRY(PTR,CHAR(*));

CALL P_ARCS(POINT,MESG);
```

The small letter n represents a user-determined value from one to 120. POINT is a pointer identifying the graph header cell of the graph the user desires to be printed. MESG is printed preceding the printing of the graph. It is printed to help the user distinguish between various print-outs.

To use P_ARCS, the user should provide a procedure called P_USER, which will be called by P_ARCS for each node and each arc of the graph pointed to by POINT. See the next chapter "User Routines" for a detailed discussion of P_USER.

3. COPY copies a graph merging in copies of graphs represented by super-nodes. The graph returned will not have any super-nodes or super-arcs. This routine is particularly useful when writing user analysis routines. Calling this routine will insure that the analysis routine will not have to traverse the interface linkage to access simple nodes if it would otherwise be necessary. The calling sequence is as follows:

```
DECLARE (PORIG,PCOPY) POINTER,
        COPY ENTRY(POINTER,POINTER);

CALL COPY(PORIG,PCOPY);
```

PORIG is a pointer identifying the graph header cell of the graph to be copies. PCOPY is a pointer identifying the graph header cell of the copy generated upon return from COPY. Calling FREEALL will return a user graph to available storage. This should be done after analysis of a copy is complete.

4. FREEALL will return a graph to available storage, that is, it will no longer exist in core storage. The calling sequence is as follows:

```
DECLARE POINT POINTER,
        FREEALL ENTRY(POINTER);

CALL FREEALL(POINT);
```

POINT should be set by the calling routine to point to the graph header cell of the graph to be returned to available storage. POINT will be returned with a NULL value.

5. DISPAGE will display on the 2250 a page of alphanumeric information. The page is set up by the user in any of his supplied routines. DISPAGE will also allow the on-line entry of 74 characters to be returned to the calling routine along with light pen and function keyboard interrupt information. A call to DISPAGE is made in the following manner:

```
DECLARE LINE(52) CHARACTER(74),
        (CODE,POS) FIXED BINARY(31),
        DISPAGE((52)CHAR(74),FIXED BIN(31),
                FIXED BIN(31));

CALL DISPAGE(LINE,CODE,POS);
```

LINE(1) is the top line displayed; LINE(2) is immediately under LINE(1) and so forth.

CODE is the LINE number (1-52) in which a cursor will be inserted. The user may then enter alphanumeric characters from the keyboard. The characters entered will replace those that were in the LINE at the time of the call to DISPAGE. If CODE is less than one or greater than 52, a cursor will not be inserted.

POS is the index (1-74) of the character in LINE(CODE) under which the cursor will be inserted. If POS is less than one or greater than 74, the cursor will not be inserted.

DISPAGE will return to the user's procedure upon either of the following actions:

a. A light pen interrupt. Upon positioning the light pen on a character displayed, CODE will be set to the LINE on which the light pen was positioned. POS will be the index (1-74) of the character in the LINE detected.

b. A function key interrupt. Upon pressing a function key, CODE will be set to a negative number or zero. The absolute value will equal the number of the key pressed.

This routine provides a straightforward manner for the user to communicate on-line with his initialization and analysis routines N_USER, ANALYZE, and BUTTONS.

V. USER ROUTINES

The following procedures should be supplied by the user prior to running the system. Two of them, N_USER and P_USER concern the initialization and printing of user-defined cell fields. ANALYZE and BUTTONS are called by the system upon an on-line user-generated interrupt.

Four fields are reserved in each NODE cell for the user. One field is reserved in each ARC cell. The user fields are declared as follows:

```
DECLARE  1 NODE BASED (PP),
           .
           .
           2 XCOR FLOAT BINARY(21),
           2 YCOR FLOAT BINARY(21),
           2 ON_OFF BIT(2),
           2 USER POINTER,
           .
         1 ARC BASED (P_ARC),
           .
           .
           2 USERA POINTER,
           .
           .
```

These fields may be initialized by the user routine N_USER and referenced by the user's analysis routines BUTTONS and ANALYZE. He may design another cell to hold additional information and set the pointer fields USER and USERA to address these cells.

User routines which reference NODE or ARC cells must contain a declaration of NODE or ARC as it appears in the program listing. If not, a compiler error message will be issued and the system will not execute. The four user routines are described below in detail.

A. N_USER

This routine is called by the system each time a node or arc is added to or removed from a graph through the Graph Manipulation Frame. The system calls N_USER in the following manner:

```
DECLARE POINT POINTER,
        IX FIXED BINARY(15),
        N_USER ENTRY(POINTER,FIXED BIN(15));

CALL N_USER(POINT,IX);
```

POINT will identify either a NODE cell or an ARC cell depending on the value of IX.

If IX equals one, the NODE cell identified by POINT will be added to the graph. Upon return from N_USER, the system will display the octagon around the corresponding node on the screen.

If IX equals two, the NODE cell identified by POINT will be deleted from the graph upon return from N_USER.

If IX equals three, the ARC cell identified by POINT will be added to the graph. Upon return from N_USER, the system will display the arrow representing the arc.

If IX equals four, the ARC cell identified by POINT will be deleted from the graph upon return from N_USER.

This routine is called before the deletion of a cell so that storage dynamically allocated for user-defined cells associated with the node and arc may be freed before the pointers to these cells are lost.

B. P_USER

This routine is called by the system routine P_ARCS. It allows the user to append text to that printed by P_ARCS for each node and arc. It is called by the system in the following manner:

```
DECLARE (ND,AR) POINTER,
        MESG CHARACTER(120),
        P_USER ENTRY(PTR,PTR,CHAR(120));

CALL P_USER(ND,AR,MESG);
```

For each node that will be printed by P_ARCS, a call to P_USER with ND identifying the NODE cell will be made. AR will be set to NULL. The user may fill the character string MESG which will be printed with the node upon return to P_ARCS.

Likewise, for each arc to be printed, another call to P_USER will be made. ND will be NULL and AR will identify the corresponding ARC cell. Again, the MESG returned will be printed with the arc.

C. BUTTONS

This routine is called by the system when the user presses a function key while the Graph Manipulation Frame is in the ready state. The call is made in the following manner:

```
DECLARE NO FIXED BINARY(15),
        BUTTONS ENTRY(FIXED BIN(15));

CALL BUTTONS(NO);
```

The variable NO will be set to the number of the function key pressed, zero through 31. By declaring the based structures and system external variables, the user has access to the graphs in the system. After analyzing a graph, the user may call system routine DISPAGE to display a page of alphanumeric information concerning the analysis and, at the same time, retrieve light pen and function key information from the 2250 on-line.

D. ANALYZE

This routine is called if the light pen is positioned on the ANALYZE option of the Graph Manipulation Frame when it is in the ready state.

The system performs the call with the statement: CALL ANALYZE. There are no calling arguments; however, by declaring the based structures and system external variables, the user has access to the graphs in the system. After analyzing a graph, the user may call system routine DISPAGE to display a page of information concerning the analysis and, at the same time, retrieve light pen and function keyboard information from the 2250 on-line.

Upon returning from ANALYZE or BUTTONS, the system will return to the Graph Manipulation Frame with the graph displayed immediately prior to the invoking of the user routine.

VI. CARD IMAGE FORMAT

A card image format is used with which graphs may be input to the system instead of using the Graph Manipulation Frame to completely specify a graph. Also, the system generates a card image file when the SAVE option is selected with the same card formats.

Four type of cards are described below. Brackets indicate that information between them is optional. Lower case letters must be replaced by entities they represent. The card format is free form in card columns one to 72. If the data shown below for a particular card will not fit on a single card, it may be continued on the next as if the card were 144 columns wide. The only places where spaces are not allowed are the following:

1) Between the characters of a node name
2) Between the characters of a graph name
3) Between the characters of the word GRAPH
4) Between the characters of the word END

The cards are organized in groups called data packets. Each packet defines a graph. The first card of a packet is the GRAPH card. It contains the name of the graph. Next node and arcs are specified by the NODE and ARC cards. Finally, an END card signifies the end of the packet. A packet may be used to augment a graph already in the system. Several NODE and ARC cards may appear in a packet. A packet may be used to augment a graph already in the system. Several NODE and ARC cards may appear in a packet interspersed. Super-nodes and super-arcs may be input along with user-defined values to be associated with the nodes and arcs.

A. GRAPH CARD

The GRAPH card starts a data packet and contains the name of the graph to be initialized or augmented.

```
GRAPH (gname)
```

"gname" is a user-defined, four-character graph name. If not specified, a default name, SSnn, will be substituted where nn is a decimal integer.

B. NODE CARD

The NODE card specifies the nodes to be added to the graph associated with the packet by the previous GRAPH card.

```
N(node specification [,node specification,
     ... ,node specification])
```

"node specification" is defined as

```
n-name[(gname)][$ user-mes $]
```

"n-name" is a user-defined, four-character node name (of which the first two characters are alphabetic).

"gname" is the graph represented by n-name if the node is a super-node. "gname" must be a graph submitted by an earlier data packet or by the user through the Graph Manipulation Frame.

"user-mes" is a maximum of forty characters passed to N_USER in the external variable MES. The dollar signs are truncated from user-mes before the call to N_USER. If a user-mes does not appear with a node, N_USER is not called.

C. ARC CARD

The ARC card specifies the arcs for the graph associated with the data packet.

```
A(arc specification [,arc specification,
     ... ,arc specification])
```

"arc specification" is defined as

```
∇-seq[$ user-mes $]
```

"∇-seq" is the sequence of characters ∇ defined in the paper. The first node named in both the starting and ending portions of the sequence are restricted to be nodes in the graph of the packet. All nodes in the sequence must be previously specified. The last node in both portions of the sequence must be a simple node.

"user-mes" is handled the same as user-mes for a node except that the N_USER parameter is set to three indicating that an arc is to be added.

D. END CARD

This card ends a data packet.

END

E. COMMENTS

Many packets (graphs) may be submitted at once. The graphs may be altered and analyzed through the Graph Manipulation Frame as if they were input through the 2250.

INTERACTIVE GRAPH REDUCTION AND ANALYSIS PROGRAM

TOTAL BUFFER ALLOCATED: NODES = 2560 ARCS = 1024 ENTIRE SYSTEM = 4096

$$$$ DATE: 06/20/70 TIME: 00:33 $$$$

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: INPUT MODE SELECTION FRAME

MESSAGE DISPLAYED: MAGNETIC DEVICE INPUT REQUESTED

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: INPUT MODE SELECTION FRAME

MESSAGE DISPLAYED: CARD INPUT REQUESTED

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: INPUT MODE SELECTION FRAME

MESSAGE DISPLAYED: BOOB TUBE INPUT REQUESTED

****

********** READY STATE **********
****

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (NONE)

OPTION SELECTED: ADD A NODE ** BY DEFAULT **

MESSAGE DISPLAYED: NEW GRAPH...ENTER ITS NAME

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (NONE)

GRAPH SS 1 INITIALIZED

MESSAGE DISPLAYED: ENTER NEW LABEL...THEN LP ON LABEL

MESSAGE DISPLAYED: OR ON _SUPER-NODE---

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (SS 1)

NODE NN 1 ADDED TO GRAPH SS 1

MESSAGE DISPLAYED: SELECT AN OPTION OR USE DEFAULTS...

****
********** READY STATE **********
****

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (SS 1)

OPTION SELECTED: ADD A NODE ** BY DEFAULT **

MESSAGE DISPLAYED: ENTER NEW LABEL...THEN LP ON LABEL

MESSAGE DISPLAYED: OR ON _SUPER-NODE---

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (SS 1)

NODE NN 2 ADDED TO GRAPH SS 1

MESSAGE DISPLAYED: SELECT AN OPTION OR USE DEFAULTS...

****
********** READY STATE **********
****

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (SS 1)

OPTION SELECTED: ADD A NODE ** BY DEFAULT **

MESSAGE DISPLAYED: ENTER NEW LABEL...THEN LP ON LABEL

MESSAGE DISPLAYED: OR ON _SUPER-NODE---

```
**********************************************

INTERRUPT SOURCE:        LIGHT PEN
FRAME DISPLAYED:  GRAPH MANIPULATION FRAME
GRAPH DISPLAYED:          (SS 1)

NODE NN 3 ADDED TO GRAPH SS 1

  MESSAGE DISPLAYED:  SELECT AN OPTION OR USE DEFAULTS...

**********************************************
**********  READY STATE  **********
**********************************************



**********************************************

INTERRUPT SOURCE:        LIGHT PEN
FRAME DISPLAYED:  GRAPH MANIPULATION FRAME
GRAPH DISPLAYED:          (SS 1)

OPTION SELECTED:  ADD AN ARC                    ** BY DEFAULT **

  MESSAGE DISPLAYED:  ADDARC...LP ON NEGATIVELY INCIDENT NODE



**********************************************

INTERRUPT SOURCE:        LIGHT PEN
FRAME DISPLAYED:  GRAPH MANIPULATION FRAME
GRAPH DISPLAYED:          (SS 1)

ARC ADDED FROM NN 1 TO NN 3 IN GRAPH SS 1

  MESSAGE DISPLAYED:  SELECT AN OPTION OR USE DEFAULTS...

**********************************************
**********  READY STATE  **********
**********************************************
```

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (SS 1)

OPTION SELECTED: ADD AN ARC ** BY DEFAULT **

MESSAGE DISPLAYED: ADDARC...LP ON NEGATIVELY INCIDENT NODE

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (SS 1)

ARC ADDED FROM NN 3 TO NN 2 IN GRAPH SS 1

MESSAGE DISPLAYED: SELECT AN OPTION OR USE DEFAULTS...

****
********** READY STATE **********
****

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (SS 1)

OPTION SELECTED: SUPER-NODE

GRAPH STORED: SS 1

MESSAGE DISPLAYED: SELECT AN OPTION OR USE DEFAULTS...

****
********** READY STATE **********
****

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (NONE)

OPTION SELECTED: ADD A NODE

MESSAGE DISPLAYED: NEW GRAPH...ENTER ITS NAME

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (NONE)

GRAPH SS 2 INITIALIZED

MESSAGE DISPLAYED: ADDNODE...LP DETECT ON "XXXX"

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (SS 2)

MESSAGE DISPLAYED: ENTER NEW LABEL...THEN LP ON LABEL

MESSAGE DISPLAYED: OR ON _SUPER-NODE---

****

INTERRUPT SOURCE: LIGHT PEN
FRAME DISPLAYED: GRAPH MANIPULATION FRAME
GRAPH DISPLAYED: (SS 2)

NODE NN 4 ADDED TO GRAPH SS 2

MESSAGE DISPLAYED: SELECT AN OPTION OR USE DEFAULTS...

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/
```


```
     /*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/
```


```
GRAPH3:  PROC OPTIONS(MAIN);

     ON ERROR CALL IHEDUMP;

     DCL   1 NODE BASED(PP),
           2 LABEL CHAR(4),
           2 COUNT FIXED DEC(2),
           2 NEXT PTR,
           2 SNODE PTR,
           2 HINFO PTR,
           2 SCREEN PTR,
           2 USER PTR,
           2 ON_OFF BIT(2),
           2 XCOR FLOAT BIN(21),
           2 YCOR FLOAT BIN(21),
           2 DOWN PTR,

     1 ARC BASED (P_ARC),
           2 SON PTR,
           2 SARC PTR,
           2 AINFO PTR,
           2 SCREAN PTR,
           2 USERA PTR,
           2 MORE PTR,
     (PP,P_ARC) PTR EXT;

     DCL
     (GSPNAME,DEVICE,MES1,KMES,COR)
           FIXED BIN(31),
     IARRAY(640) FIXED BIN(31),
     (NUL,MES2,KM2,G2,CSEQ,CL,CN,ATTEN,CODE,I10(10),FKEY(11),G3,KNAME)
           FIXED BIN(31) EXT,
     (KM22,KEY3,KEY2,TOTAL2,TOTAL3,IC(3),G1,CHOICE,C_ALL)
           FIXED BIN(31) EXT,
     UNIT FIXED BIN(31) INITIAL(10),
     (X,Y) FLOAT BIN(21),
     (NX(8),NY(8))
           FLOAT BIN(21) EXT,
     (RS,PNN) PTR,
     (TOP,FIRST,PN,SUPLIST,PF,SN) PTR EXT,
     FUNCTION(11) LABEL,
     FOUND(3) LABEL,
     (ALPHA,FRAME) CHAR(26) EXT,
     NUMERIC CHAR(10) EXT,
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM      *****/

     DAT CHAR(6) EXT,
     TIM CHAR(9) EXT,
     OUT CHAR(132)VAR EXT,
     MES CHAR(40) EXT,
     ICT PICTURE 'Z9',
     (SNDCT,NDCT) FIXED DEC(2,0) EXT,
     ISW BIT(1) EXT;

     DCL          /*****   ENTRY POINTS   *****/
     ENATN ENTRY(,FIXED BIN(31),FIXED BIN(31),FIXED BIN(31)),
     GSPRD ENTRY  (,,FIXED BIN(31),FIXED BIN(31),),
     ICURS ENTRY(,,,FIXED BIN(31)),
     INGDS ENTRY(,,FIXED BIN(31),,),
     MLITS ENTRY(,FIXED BIN(31)),
     MLPEO ENTRY(,FIXED BIN(31), FIXED BIN(31)),
     ORGEN ENTRY(FIXED BIN(31),FIXED BIN(31),FIXED BIN(31)),
     PLINE ENTRY(,,,,,FIXED BIN(31),FIXED BIN(31)),
     PTEXT ENTRY(,,FIXED BIN(31),,,FIXED BIN(31),FLOAT BIN(21),FLOAT BIN
     (21)),
     SCHAM ENTRY(,FIXED BIN(31)),
     SDATL ENTRY(,FLOAT BIN(21),FLOAT BIN(21),FLOAT BIN(21),FLOAT BIN
          (21)),
     SGDSL ENTRY(,FLOAT BIN(21),FLOAT BIN(21),FLOAT BIN(21),
                 FLOAT BIN(21),FLOAT BIN(21),FLOAT BIN(21),
                 FLOAT BIN(21),FLOAT BIN(21)),
     SLPAT ENTRY(,FIXED BIN(31)),
     SSCIS ENTRY(,FIXED BIN(31)),
     STPOS ENTRY(,FLOAT BIN(21),FLOAT BIN(21)),
     ALLOC ENTRY(FIXED BIN,PTR),
     CLEANUP ENTRY(CHAR(*)VAR),
     CLN ENTRY(CHAR(*)) RETURNS(CHAR(132)VAR),
     TIMER ENTRY(FIXED BIN),
     UPDISP2 ENTRY(PTR,FIXED BIN);

   /*****   INITIALIZE NULL VARIABLE AND CORRELATION VARIABLES   *****/
     ALPHA = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
     NUMERIC = '0123456789';
     OPEN FILE(SYSPRINT) PAGESIZE(48);
     ISW = '1'B;
     NUL = 5;
     IC(1) = -1;
     IC(2) = -2;
     IC(3) = -3;
     DAT = DATE;
     N1 = 10;
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM      *****/

   /*****   INITIALIZE GSP AND DEVICE    *****/
     CALL INGSP(GSPNAME,NUL);
     CALL INDEV(GSPNAME,UNIT,DEVICE);
          CALL TMDEV(DEVICE);
          CALL TMGSP(GSPNAME);
     NUL = 5;
     UNIT = 10;
     CALL INGSP(GSPNAME,NUL);
     CALL INDEV(GSPNAME,UNIT,DEVICE);

   /*****   CREATE AN ATTENTION LEVEL    *****/
     CALL CRATL(DEVICE,ATTEN);
     CALL ENATN(ATTEN,0,-31,34);
     CALL MLITS(ATTEN,3);
     CALL MLPEO(ATTEN,2,2);

     BEGINNING:
     SNDCT, NDCT = 0;
     TOP = NULL;
     SUPLIST = NULL;
     TOTAL2 = N1 * 256;   TOTAL3 = (14-N1) * 256;
     PUT LIST(REPEAT(' ',10)||
              'INTERACTIVE GRAPH REDUCTION AND ANALYSIS PROGRAM')
              LINE(10);
     PUT LIST(CLN('TOTAL BUFFER ALLOCATED:  NODES = '||TOTAL2||
     '    ARCS = '||TOTAL3||'   ENTIRE SYSTEM = '||(TOTAL2+TOTAL3+
          512)))SKIP(3);


   /*****   INITIALIZE GRAPHIC DATA SET:  G1 AND G2   *****/
     CALL INITIAL;

INITIAL:  PROC;
     CALL INGDS(DEVICE,G1,TOTAL2  ,NUL,G2);
     CALL SSCIS(G1,1);
     CALL SLPAT(G1,1);

   /*****   INITIALIZE GRAPHIC DATA SET:  MES1    *****/
   /*****                                 MES2    *****/
     CALL INGDS(DEVICE,MES1,NUL,NUL,MES2);
     CALL SCHAM(MES1,2);
     CALL SSCIS(MES1,1);
     CALL SDATL(MES1,.5,.5,49.5,1.5);
     CALL SGDSL(MES1,0.,5.,49.,6.,0.,0.,49.,35.);

   /*****   PLOT TEXT IN 'G1'    *****/
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/

     CALL INITG1;

   /*****   GENERATE G2   *****/

     CALL SGDSL(G2,66.,0.,74.,52.,0.,0.,74.,52.);
     CALL SDATL(G2,.5,52.5,8.5,.5);
     CALL SCHAM(G2,3);
     CALL SSCIS(G2,1);
     CALL SGRAM(G2,1);
     CALL SLPAT(G2,1);

     CALL STPOS(G2,1.,1.);
     CALL PTEXT(G2,'OPTIONS:',8,NUL,NUL,NUL,1., 4.);
     CALL PTEXT(G2,'ADD-'    ,4,NUL,NUL,NUL,1., 6.);
     CALL PTEXT(G2,'SUPER'   ,5,NUL,NUL,NUL,1.,10.);
     CALL PTEXT(G2,'MOVE A'  ,6,NUL,NUL,NUL,1.,13.);
     CALL PTEXT(G2,'REMOVE'  ,6,NUL,NUL,NUL,1.,16.);
     CALL PTEXT(G2,'_NODE'   ,5,NUL,FKEY(1),1,4., 7.);
     CALL PTEXT(G2,'_ARC'    ,4,NUL,FKEY(2),1,4., 8.);
     CALL PTEXT(G2,'_NODE'   ,5,NUL,FKEY(3),1,4.,11.);
     CALL PTEXT(G2,'_NODE'   ,5,NUL,FKEY(4),1,4.,14.);
     CALL PTEXT(G2,'_NODE'   ,5,NUL,FKEY(5),1,4.,17.);
     CALL PTEXT(G2,'_ARC'    ,4,NUL,FKEY(6),1,4.,18.);
     CALL PTEXT(G2,'_CANCEL' ,7,NUL,FKEY(7),1,1.,20.);
     CALL PTEXT(G2,'_SAVE'   ,5,NUL,FKEY(8),1,1.,22.);
     CALL PTEXT(G2,'_DISPLAY',8,NUL,FKEY(9),1,1.,24.);
     CALL PTEXT(G2,'_ANALYZE',8,NUL,FKEY(10),1,1.,26.);
     CALL PTEXT(G2,'_ALL',4,NUL,C_ALL,1,1.,28.);
     CALL PTEXT(G2,'_QUIT'   ,5,NUL,FKEY(11),1,1.,30.);
     CALL PTEXT(G2,'NAME',4,NUL,NUL,1,2., 1.);
     CALL PTEXT(G2,'____',4,NUL, KNAME,1,2., 2.);

   /*****   GENERATE 49 GRAPHIC NODES
            THEY ARE USED TO LOGICALLY
            CONNECT A USER'S DIRECTED
            GRAPH TO THE GEOMETRIC REALIZATION
                  DISPLAYED                  *****/

     CALL SGDSL(G2, 0.,228.,1820.,2048.,0.,0.,2048.,2048.);
     CALL SDATL(G2,0.,1820.,1820.,0.);

     CALL ALLOC(1,FIRST);
     FIRST -> LABEL = '$FR$';
     FIRST -> NEXT, FIRST -> HINFO = FIRST;
     ICT = 0;
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/

      DO X = 70. TO 1750. BY 280.;
      DO Y = 70. TO 1750. BY 280.;

      CALL ALLOC(1,PN);
      ICT = ICT + 1;
      PN -> LABEL = ICT || 'XX';
      PN -> XCOR = X;
      PN -> YCOR = Y;

      PF = FIRST -> NEXT;
      PN -> NEXT = PF;
      PN -> HINFO = FIRST;
      PF -> HINFO, FIRST -> NEXT = PN;

    /*****   CL.....CORRELATION VALUE FOR THE LABEL     *****/
      UNSPEC(CL) = UNSPEC(PN);
      CL = ABS(CL);
      CALL STPOS(G2,X-50.,Y);
      CALL PTEXT(G2,'XXXX'   ,4,CL,NUL,1,X-50.,Y);

      END; END;

    /*****   PLOT TEXT IN 'MES1'   *****/
      CALL STPOS(MES1,1.,1.);
      CALL GETTIME;
      OUT = '$$$$ '||MES||' $$$$';
      PUT LIST(REPEAT(' ',15)||CLN(OUT))SKIP(2);
      CALL PTEXT(MES1,REPEAT(' ',6)||SUBSTR(MES,1,33),40,NUL,KMES,
           1,1.,1.);
      CALL PTEXT(MES1,MES,40,NUL,KMES,1,6.,1.);
      CALL PTEXT(MES1,REPEAT('*',39),40,NUL,NUL,1,1.,2.);

    /*****   PLOT TEXT IN 'MES2'   *****/
      CALL SCHAM(MES2,1);
      CALL SSCIS(MES2,1);
      CALL SDATL(MES2,.5,5.5,74.5,.5);
      CALL SGDSL(MES2,0.,0.,74.,5.,0.,0.,74.,52.);

GETTIME:  PROC;
      TIM = TIME;
      OUT = 'DATE:  '||SUBSTR(DAT,3,2)||'/'||SUBSTR(DAT,5)||'/'||
            SUBSTR(DAT,1,2)||'    TIME:  '||SUBSTR(TIM,1,2)||':'||
            SUBSTR(TIM,3,2);
      MES = '      '||CLN(OUT);
      END GETTIME;
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/

     MES = 'SELECT AN OPTION OR USE DEFAULTS...';
     CALL STPOS(MES2,0.,1.);
     CALL PTEXT(MES2,MES,40,NUL,KM2,1,1.,3.);
     CALL PTEXT(MES2,REPEAT('*',39),40,NUL,KM22,1,1.,5.);

   /*****   INITIALIZE G3 FOR ARCS   *****/
     CALL INGDS(DEVICE,G3,TOTAL3,NUL,NUL);
     CALL SGDSL(G3, 0.,228.,1820.,2048.,0.,0.,2048.,2048.);
     CALL SDATL(G3,0.,1820.,1820.,0.);
     CALL SCHAM(G3,3);
     CALL SSCIS(G3,1);
     CALL SGRAM(G3,1);
     CALL SLPAT(G3,1);

   /*****   DRAW A BOX AROUNT THE GRAPH DISPLAY AREA   *****/
     NX(1), NX(4), NY(3), NY(4) = 1.;
     NX(2), NX(3), NY(1), NY(2) = 1819.;
     CALL STPOS(G3,1.,1.);
     CALL PLINE(G3,NX(1),NY(1),NUL,NUL,1,4);

     END INITIAL;

   /*****   DISPLAY 'G1'   *****/
     DISPLAY_G1:
     CALL SALRM(DEVICE);
     CALL EXEC(MES1);
     CALL EXEC(G1);
     CALL OMIT(G3);
     FRAME = 'INPUT MODE SELECTION FRAME';

     ATTENT:        /*****   REQUEST ATTENTION INFORMATION     *****/
     IF FRAME = 'GRAPH MANIPULATION FRAME' THEN DO;
     PUT LIST(REPEAT('*',49))SKIP(3);
     PUT LIST(REPEAT('*',18)||' READY STATE '||REPEAT('*',17))SKIP;
     PUT LIST(REPEAT('*',49))SKIP;
     END;
     CALL RQ;

     IF CODE ¬= 34 THEN DO;
          CALL BUTTONS;
          GO TO START_OVER;
          END;

   /*****   RESPOND TO LIGHT PEN ATTENTION   *****/
     IF I10(1) = G1 THEN DO;
          DO I = 1 TO 3;
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/

          IF I10(4) = IC(I) THEN DO;
               Y = 6. + I;
               CALL STPOS(G1,7.,Y);
               CALL PTEXT(G1,'X',1,NUL,CHOICE,3,7.,Y);
               GO TO FOUND(I);
               END;
          END;

     MES = 'INVALID OPTION SELECTED.  TRY AGAIN';
     GO TO RESTRT;
     END;


     IF I10(1) = G2 THEN DO;
   /*****   CHECK FOR GRAPHIC OPTION SELECTION:  G2   *****/
          DO I = 1 TO 11;
          IF I10(2) = FKEY(I) THEN DO;
               IF I < 7 THEN GO TO FUNCTION(I);
               IF I = 11 THEN GO TO DISPLAY_G1;
               MES = 'OPTION SELECTED IS NOT SUPPORTED...';
               CALL UPMES2;
               MES = 'AT THIS TIME---SELECT ANOTHER';
               CALL UPMES2;
               CALL TIMER(2);
               GO TO START_OVER;
               END;
          END;

   /*****   DEFAULT USED, CHECK FOR DETECTION OF NODE   *****/
     IF I10(4) ¬= 0 THEN DO;
          COR = ABS(I10(4));
          UNSPEC(PN) = UNSPEC(COR);
          IF I10(4) > 0
               THEN IF ¬SUBSTR(PN->ON_OFF,2,1)
                    THEN CALL ADDNODE(PN);
                  ELSE IF PN->ON_OFF = '11'B
                       THEN CALL ADDARC(PN);
                       ELSE;

   /*****   DETECT ON NODE   *****/
               ELSE CALL ADDARC(PN);
          GO TO START_OVER;
          END;

     MES = 'INVALID OPTION---TRY AGAIN.';
     GO TO RESTRT_MES2;
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/
     END;

   /*****   DEFAULT USED, CHECK FOR DETECTION OF AN ARC   *****/
     IF I10(1) = G3 THEN DO;
          IF I10(4) = 0 THEN GO TO START_OVER;
          UNSPEC(PN) = UNSPEC(I10(4));
          CALL RMVARC(PN);
          GO TO START_OVER;
          END;

     CALL BUTTONS;
     GO TO START_OVER;

     FUNCTION(1):    /*****  ADD A NODE   *****/
     PN = NULL;
     CALL ADDNODE(PN);
     GO TO START_OVER;

     FUNCTION(2):     /*****   ADD AN ARC   *****/
     PN = NULL;
     CALL ADDARC(PN);
     GO TO START_OVER;

     FUNCTION(3):    /*****   SUPER NODE   *****/
     CALL SUPNODE;
     GO TO START_OVER;

     FUNCTION(4):      /*****   MOVE A NODE   *****/
     CALL MOVENOD;
     GO TO START_OVER;


     FUNCTION(5):    /*****   REMOVE A NODE   *****/
     CALL RMVNODE;
     GO TO START_OVER;

     FUNCTION(6):    /*****   REMOVE AN ARC   *****/
     PN = NULL;
     CALL RMVARC(PN);
     GO TO START_OVER;

     FOUND(1):
     CALL STPOS(G1,7o,7o);
     CALL PTEXT(G1,' ',1,NUL,CHOICE,3,7o,7o);
     CALL STPOS(MES1,1o,1o);
     CALL PTEXT(MES1,REPEAT(' ',39),40,NUL,KMES,3,1o,1o);
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/

      CALL EXEC(MES2);
      CALL EXEC(G2);
      CALL INCL(G3);
      CALL EXEC(G3);
      PUT LIST('MESSAGE DISPLAYED:  BOOB TUBE INPUT REQUESTED')SKIP(2);
      FRAME = 'GRAPH MANIPULATION FRAME';
      GO TO ATTENT;

      FOUND(2):
      MES = 'CARD INPUT REQUESTED';
      GO TO RESTRT;

      FOUND(3):
      MES = 'MAGNETIC DEVICE INPUT REQUESTED';

      RESTRT:
      PUT LIST('MESSAGE DISPLAYED:  '||MES)SKIP(2);

    /*****   UPDATE TEXT IN 'MES1'   *****/
      CALL EXEC(MES1);
      CALL EXEC(G1);
      CALL OMIT(G3);
      CALL STPOS(MES1,1o,1o);
      CALL PTEXT(MES1,MES,40,NUL,KMES,3,1o,1o);
      CALL EXEC(MES1);
      GO TO ATTENT;

      START_OVER:
      CALL UPDISP2(FIRST,1);
      MES = 'SELECT AN OPTION OR USE DEFAULTSooo';

      RESTRT_MES2:
      CALL UPMES2;
      GO TO ATTENT;

      TERMINATE:
           PUT LIST('TERMINATE ALLooooo')SKIP(3);
           CALL TMGDS(G1);
           CALL TMGDS(G3);
           CALL TMGDS(MES1);
           CALL TMDEV(DEVICE);
           CALL TMGSP(GSPNAME);
BUTTONS:   PROC;
    /*****   RESPOND TO FUNCTION BUTTON 1   *****/
      IF CODE = 1 THEN GO TO TERMINATE;
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/

   /*****   RESPOND TO FUNCTION BUTTON 3   *****/
     IF CODE = 3 THEN DO;
          CALL TMGDS(MES1);
          CALL TMGDS(G1);
          CALL TMGDS(G3);
          CALL FREEALL(TOP);
          GO TO BEGINNING;
          END;

   /*****   RESPOND TO FUNCTION BUTTON 4   *****/
     IF CODE = 4 THEN DO;  CALL SUPDUMP;   RETURN;   END;

   /*****   RESPOND TO FUNCTION BUTTON 5   *****/
     IF CODE = 5 THEN GO TO DISPLAY_G1;

     MES = 'UNRECOGNIZED ATTENTION SOURCE...TRY AGAIN';
     GO TO RESTRT;
     END BUTTONS;

SUPDUMP:  PROC;
          MES = '...DUMP IN PROCESS...';
          CALL UPMES2;
          PF = SUPLIST;
          DO WHILE(PF¬=NULL);
          CALL P_ARCS(PF -> SON,'SUPER DUMP');
          PF = PF -> MORE;
          END;

     PUT LIST('SYSTEM DEBUG ROUTINE SUPDUMP INVOKED')SKIP(3);
          IF TOP ¬= NULL THEN PUT LIST('TOP -> LABEL = '||
               TOP->LABEL)SKIP(3);
          I = 1;
          PNN = SUPLIST;
          DO WHILE('1'B);

          IF I = 1 THEN DO;
               PF = FIRST;
               I = 2;
               END;

               ELSE DO;
               PF = PNN;
          IF PF = NULL THEN DO;
          PUT LIST('****  PF = NULL--NO DUMP...')SKIP(2);
          I = 3;
          GO TO END_DO;
```

```
END;
     PNN = PNN -> MORE:
     PF = PF -> SON;
     END;
PUT LIST('DUMP OF NODES   '||PF -> LABEL)SKIP(2);
PN = PF;
DO WHILE('1'B);
OUT = PN->LABEL||'  '||PN->ON_OFF;
IF PF = FIRST THEN DO;
     IF PN->ON_OFF = '00'B & PN->SNODE = NULL &
          PN->DOWN = NULL THEN GO TO SKIP_IT;
     RS = PN->SNODE;
     IF RS ¬= NULL THEN
     OUT = OUT||'('||RS->LABEL||')';
     ELSE OUT = OUT||'('||'----'||')';
     END;
ELSE IF PN->SCREEN = NULL THEN OUT = OUT||'(----)';
     ELSE DO;
          SN = PN->SCREEN;
          OUT = OUT || '('  ||  SN->LABEL||')';
          END;
SN = PN -> DOWN;
DO WHILE(SN ¬= NULL);
     RS = SN -> SON;
     OUT = OUT||'  -->  '||RS -> LABEL||','||
          DECIMAL(UNSPEC(SN));
     IF SN -> SARC = NULL THEN OUT = OUT||'(0)';
          ELSE OUT = OUT||'(1)';
     IF SN->AINFO ¬= PN THEN OUT = OUT||'**'||
          PN->LABEL||'**';
     CALL CLEANUP(OUT);
     SN = SN -> MORE;
     END;
CALL CLEANUP(OUT);
PUT LIST(OUT)SKIP;
SKIP_IT:
PN = PN -> NEXT;
IF PN = PF THEN GO TO END_DO;
END;

END_DO:
IF I = 3 THEN RETURN;
END;
END SUPDUMP;
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/

     /*****     RETURNS PTR TO SUPNODE HEADER CELL     *****/
SELSUP:  PROC(PN);
     DCL (PN,SN)PTR,
          (TRMCD,KEYMORE,KEYEND)     FIXED BIN(31),
     END BIT(1),
     Y FLOAT BIN(21);
     MES = 'FROM SELSUP ***  CALL ACKNOWLEDGED...';
     CALL UPMES2;
     CALL TIMER(1);

     FRAME = 'GRAPH SELECTION FRAME';
     TRMCD = 1;
     END = '0'B;
     NO = 19;
     KEYMORE, KEYEND = -1;
     PN = SUPLIST;
     IF PN = NULL THEN RETURN;
     CALL OMIT(G3);
     CALL EXEC(G3);
     MES = 'HERE COMES THE GSPRD...';   CALL UPMES2;   CALL TIMER(2);
     CN = 2560;  CL = 2;     CALL GSPRD(G2,IARRAY(1),CN,CL,TRMCD);
  /*
     CALL GSPRD(G2,IARRAY(1),2560,2,TRMCD);
  */
     CALL SGDSL(G2,31.,0.,36.,21.,0.,0.,74.,52.);
     CALL SDATL(G2,.5,21.5,5.5,.5);
     PUT LIST('TRMCD = '||TRMCD)SKIP(2);
     PUT LIST('DUMMY LINE.....')SKIP(2);

     ONCEMORE:
     MES = 'LP ON SUPERNODE DESIRED OR _ALL...';
     CALL UPMES2;
     MES = 'OR ON _CANCEL OR *END* OR *MORE ---';
     CALL UPMES2;

     AGAIN:
     I = 0;
     DO WHILE(I<NO & ¬END);
          CALL RESET(G2);
          IF PN -> SON = TOP THEN GO TO NEXTNO;
          I = I + 1;
          Y = I;
          SN = PN -> SON;
          CL = BINARY(UNSPEC(SN));
          PUT LIST('SELSUP...VISITING SUPERNODE = '||
               DECIMAL(UNSPEC(SN))||' = '||SN->LABEL)SKIP(2);
```

```
        CALL STPOS(G2,1o,Y);
        CALL PTEXT(G2,SN->LABEL||' ',5,CL,NUL,1  ,1o,Y);

        NEXTND:
        PN = PN->MORE;
        IF PN ¬= NULL THEN GO TO ENDDO;
        CALL STPOS(G2,1o,Y+2o);
        CALL PTEXT(G2,'*END ',5,NUL,KEYEND,1  ,1o,Y+2o);
       PUT LIST('*END OF LIST')SKIP(2);
        END = '1'B;
        GO TO SELECT;

        ENDDO:
        END;

Y = I + 2o;
CALL STPOS(G2,1o,Y);
CALL PTEXT(G2,'*MORE',5,NUL,KEYMORE,1,1o,Y);
PUT LIST('*MORE TO COME')SKIP(2);

SELECT:
CALL EXEC(G2);
CALL RQ;
IF CODE ¬= 34 THEN GO TO ABEND;
IF I10(1) ¬= G2 THEN GO TO ERROR;
IF I10(4) ¬= 0 THEN GO TO SEL2;
IF I10(2) = KEYEND THEN GO TO ABEND;
IF I10(2) =KEYMORE THEN GO TO AGAIN;

SEL2:
UNSPEC(PN) = UNSPEC(ABS(I10(4)));

SEL3:
CALL RESET(G2);
CALL SGDSL(G2, 0o,228o,1820o,2048o,0o,0o,2048o,2048o);
CALL SDATL(G2,0o,1820o,1820o,0o);
CALL ORGEN(G2,IARRAY(1),TRMCD);
CALL EXEC(G2);
CALL INCL(G3);
CALL EXEC(G3);
RETURN;

ABEND:
PN = NULL;
GO TO SEL3;
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM      *****/

      ERROR:
      MES = 'INVALID LP DETECT...TRY AGAIN';
      CALL UPMES2;
      CALL TIMER(1);
      GO TO ONCEMORE;
      END SELSUP;

SUPNODE:  PROC;
      DCL CA FIXED BIN(31);
      PUT LIST('OPTION SELECTED:  SUPER-NODE')SKIP(2);
      IF TOP = NULL THEN DO;
           PUT LIST('NO GRAPH DISPLAYED AT TIME OF CALL')SKIP(2);
           RETURN;
           END;

      ELSE PUT LIST('GRAPH STORED:  '||TOP->LABEL)SKIP(2);


      RS = TOP->NEXT;
      DO WHILE(RS¬=TOP);
           PN = RS->SCREEN;
           CL = ABS(BINARY(UNSPEC(PN)));
           CN = -CL;
           CALL OMIT(G2,CN);
           PN->SNODE = NULL;
           PN->ON_OFF = '10'B;

           CALL STPOS(G2,PN->XCOR-50.,PN->YCOR);
           CALL PTEXT(G2,'XXXX',4,CL,NUL,3,PN->XCOR-50.,PN->YCOR);
           SN = RS -> DOWN;
           DO WHILE(SN¬=NULL);
                SNN = SN -> SCREAN;
                SNN -> SARC = NULL;
                CA = ABS(BINARY(UNSPEC(SNN)));
                CALL OMIT(G3,CA);
                SN = SN->MORE;
                END;
           RS = RS -> NEXT;
           END;

      TOP = NULL;
      CALL SGDSL(G2,66.,0.,74.,52.,0.,0.,74.,52.);
      CALL SDATL(G2,.5,52.5,8.5,.5);

      CALL STPOS(G2,2.,2.);
      CALL PTEXT(G2,'____',4,NUL,KNAME,3,2.,2.);
```

```
/*****     GRAPH REDUCTION AND ANALYSIS PROGRAM     *****/

     CALL SGDSL(G2, 0o,228o,1820o,2048o,0o,0o,2048o,2048o);
     CALL SDATL(G2,0o,1820o,1820o,0o);

     END SUPNODE;

FREEALL:  PROC(TOP);
     DCL (TOP,PN) PTR;

     IF TOP = NULL THEN RETURN;
     PUT LIST('SYSTEM ROUTINE FREEALL INVOKED')SKIP(2);
     OUT = 'STORAGE FOR GRAPH '||TOP->LABEL||' SUCCESSFULLY FREED';

     SN = TOP -> DOWN;
     TOP -> DOWN = NULL;
     DO WHILE(SN ¬= NULL);
          CALL FREEARC(SN->AINFO);
          PN = SN;
          SN = SN -> MORE;
          FREE PN -> ARC;
          END;

     PN = TOP -> NEXT;
     TOP -> NEXT = NULL;
          DO WHILE(PN¬=NULL);
          SN = PN -> DOWN;
          CALL FREEARC(SN);
               PF = PN;
               PN = PN -> NEXT;
               FREE PF -> NODE;
               END;
               PUT LIST(CLN(OUT))SKIP(2);
     END FREEALL;

FREEARC:  PROC(SN);
     DCL (SN,SF) PTR;
          DO WHILE(SN ¬= NULL);
               SF = SN;
               SN = SN -> MORE;
               FREE SF -> ARC;
               END;
     END FREEARC;

     END GRAPH3;
```

```
/*****      ADD AN ARC      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72)');
ADDARC:  PROC(PM);
      DCL   1 NODE BASED(PP),
              2 LABEL CHAR(4),
              2 COUNT FIXED DEC(2),
              2 NEXT PTR,
              2 SNODE PTR,
              2 HINFO PTR,
              2 SCREEN PTR,
              2 USER PTR,
              2 ON_OFF BIT(2),
              2 XCOR FLOAT BIN(21),
              2 YCOR FLOAT BIN(21),
              2 DOWN PTR,

      1 ARC BASED (P_ARC),
              2 SON PTR,
              2 SARC PTR,
              2 AINFO PTR,
              2 SCREAN PTR,
              2 USERA PTR,
              2 MORE PTR,
      (PP,P_ARC) PTR EXT;
      DCL (SM,PN,PM) PTR,
      TIMER ENTRY(FIXED BIN),
      UPDISP2 ENTRY(,FIXED BIN),
      ALLOC ENTRY(FIXED BIN,PTR),
      DISPARC ENTRY(,,),
      MES CHAR(40) EXT,
      (PF,SN,TOP,FIRST)PTR EXT,
      (CODE,I10(10),FKEY(11),G2,CN,G3,TOTAL3,KEY3)FIXED BIN(31) EXT;

      PUT LIST('OPTION SELECTED:  ADD AN ARC')SKIP(2);
      IF TOP = NULL THEN GO TO ERROR2;
      SN = TOP->NEXT;    IF SN = TOP THEN GO TO ERROR2;
      SN = SN->NEXT;     IF SN = TOP THEN GO TO ERROR2;

      PN = PM;
      IF PN ¬= NULL THEN DO;
            PUT LIST('** BY DEFAULT **');
            GO TO DEFAULT;
            END;
      CALL UPDISP2(FIRST,2);

      L2:
      MES = 'ADDARC...LP ON POSITIVELY INCIDENT NODE';
```

```
/*****     ADD AN ARC     *****/

     CALL UPMES2;
     CALL RQ;
     IF CODE ¬= 34 THEN RETURN;
     IF I10(2) = FKEY(7) THEN RETURN;
     IF I10(1) ¬= G2 THEN GO TO ERROR1;
     IF I10(4) = 0 THEN GO TO ERROR1;
     UNSPEC(PN) = UNSPEC(ABS(I10(4)));
     IF PN -> ON_OFF ¬= '11'B THEN GO TO ERROR1;

     DEFAULT:
     MES = 'ADDARC...LP ON NEGATIVELY INCIDENT NODE';
     L3:
     CALL UPMES2;
     CALL RQ;
     IF CODE ¬= 34 THEN RETURN;

     IF I10(2) = FKEY(7) THEN RETURN;
     IF I10(1) ¬= G2 THEN GO TO SCREW_UP;
     IF I10(4)  = 0 THEN GO TO SCREW_UP;
          UNSPEC(PF) = UNSPEC(ABS(I10(4)));
          IF PF = PN THEN GO TO NO_LOOPS:
          IF PF->ON_OFF ¬= '11'B THEN GO TO SCREW_UP;
          SN = PN -> DOWN;
          DO WHILE(SN¬=NULL);
          IF SN -> SON = PF & SN -> SARC = NULL THEN DO;
               CN = BINARY(UNSPEC(SN));
               CALL INCL(G2,CN);
               GO TO CONTINUE;
               END;
          SN = SN -> MORE;
          END;

     IF SN = NULL THEN DO:
          CALL ALLOC(2,SN);
          SN -> SON = PF;
          CALL ADARCBT(PN,SN);
          CALL DISPARC(PN,PF,SN);
          END:

     CONTINUE:
     SM = SN;
     SM -> AINFO = PN;
     PN = PN -> SNODE;
     PF = PF -> SNODE;
     CALL ALLOC(2,SN);
     SN -> SON = PF;
```

```
/*****      ADD AN ARC      *****/

      SN -> SCREAN = SM;
      SM -> SARC = SN;
      PF -> COUNT = PF -> COUNT + 1;

   /*****
         IF PF -> SNODE ¬= NULL | PN -> SNODE ¬= NULL
              THEN CALL SUPCON(PN,PF,SN);
                   *****/
         CALL ADARCBT(PN,SN);
         PUT LIST('ARC ADDED FROM '||PN->LABEL||' TO '||PF->LABEL||
              ' IN GRAPH '||TOP->LABEL)SKIP(2);
   /*    CALL N_USER(SN,3);    */
         RETURN;

      SCREW_UP:
      MES = 'INVALID NODE DETECTED---TRY AGAIN';
      CALL UPMES2;
      GO TO DEFAULT;

      ERROR1:
      MES = 'INVALID LP DETECT---TRY AGAIN';
      CALL UPMES2;
      GO TO L2;

      ERROR2:
         MES = 'LP ON _CANCEL...INSUFFICIENT NUMBER';
         CALL UPMES2;
         MES = 'OF NODES ARE DISPLAYED---';
         GO TO L3;

      NO_LOOPS:
      MES = 'SORRY---NO LOOPS ALLOWED..';
      CALL UPMES2;
      CALL TIMER(2);
      END ADDARC;
```

```
/*****      ADD A NODE      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72)');
ADDNODE:  PROC(PN);
      DCL   1 NODE BASED(PP),
              2 LABEL CHAR(4),
              2 COUNT FIXED DEC(2),
              2 NEXT PTR,
              2 SNODE PTR,
              2 HINFO PTR,
              2 SCREEN PTR,
              2 USER PTR,
              2 ON_OFF BIT(2),
              2 XCOR FLOAT BIN(21),
              2 YCOR FLOAT BIN(21),
              2 DOWN PTR,
      1 ARC BASED (P_ARC),
              2 SON PTR,
              2 SARC PTR,
              2 AINFO PTR,
              2 SCREAN PTR,
              2 USERA PTR,
              2 MORE PTR,
      (PP,P_ARC) PTR EXT;

      DCL (PF,RS,PN,NN)PTR,
      LB CHAR(4),
      (CODE,I10(10),FKEY(11),CL,G2,NUL,CN,KNAME) FIXED BIN(31) EXT,
      (TOTAL2,KEY2) FIXED BIN(31) EXT,
      MES CHAR(40)EXT,
      (TOP,FIRST,SUPLIST) PTR EXT,
      ALPHA CHAR(26) EXT,
      UPDISP2 ENTRY(,FIXED BIN),
      TCURS ENTRY(,,,FIXED BIN(31)),
      GSPRD ENTRY(,,FIXED BIN(31),FIXED BIN(31),,,),
      STPOS ENTRY(,FLOAT BIN(21),FLOAT BIN(21)),
      PTEXT ENTRY(,,FIXED BIN(31),,,FIXED BIN(31),FLOAT BIN(21),
            FLOAT BIN(21)),
      ALLOC ENTRY(FIXED BIN,PTR),
      DISPNOD ENTRY(FLOAT BIN(21),FLOAT BIN(21),PTR),
      SGDSL ENTRY(,FLOAT BIN(21),FLOAT BIN(21),FLOAT BIN(21),
                  FLOAT BIN(21),FLOAT BIN(21),FLOAT BIN(21),
                  FLOAT BIN(21),FLOAT BIN(21)),
      SDATL ENTRY(,FLOAT BIN(21),FLOAT BIN(21),
                  FLOAT BIN(21),FLOAT BIN(21)),
      (SNDCT,NDCT) FIXED DEC(2,0) EXT,
      TERMCODE FIXED BIN(31);
```

```
/*****      ADD A NODE      *****/

      PUT LIST('OPTION SELECTED:  ADD A NODE')SKIP(2);
      IF PN ¬= NULL THEN PUT LIST('** BY DEFAULT **');
      IF TOP ¬= NULL THEN GO TO DEFINED;
      CALL SGDSL(G2,66.,0.,74.,52.,0.,0.,74.,52.);
      CALL SDATL(G2,.5,52.5,8.5,.5);

      MES = 'NEW GRAPH...ENTER ITS NAME';
      CALL UPMES2;
      CALL ICURS(G2,NUL,KNAME,1);
      CALL EXEC(G2);
      CALL RQ;
      CALL RCURS(G2);
      CALL GSPRD(G2,LB,4,1,TERMCODE,NUL,KNAME);
      IF LB = '    ' THEN DO;
           SNDCT = SNDCT + 1;
           LB = 'SS'||SUBSTR(CHAR(SNDCT),4);
           END;
      CALL STPOS(G2,2., 2.);
      CALL PTEXT(G2,LB,4,NUL,KNAME,3,2., 2.);
      CALL EXEC(G2);
      CALL ALLOC(1,TOP);
      TOP -> LABEL = LB;
      TOP -> NEXT = TOP;
      CALL ALLOC(2,PF);
      PF -> MORE =  SUPLIST;
      SUPLIST = PF;
      PF -> SON = TOP;
      PUT LIST('GRAPH '||TOP->LABEL||' INITIALIZED')SKIP(2);
      CALL SGDSL(G2, 0.,228.,1820.,2048.,0.,0.,2048.,2048.);
      CALL SDATL(G2,0.,1820.,1820.,0.);

      DEFINED:
      IF PN ¬= NULL THEN GO TO DEFAULT;
      CALL UPDISP2(FIRST,1);
      MES = 'ADDNODE...LP DETECT ON "XXXX"';
      CALL UPMES2;

      AGAIN:
      CALL RQ;
      IF CODE ¬= 34 THEN RETURN;
      IF II0(1) ¬= G2 THEN GO TO ERROR;
      IF II0(2) = FKEY(7) THEN RETURN;
      IF II0(4) <= 0 THEN GO TO ERROR;
      UNSPEC(PN) = UNSPEC(II0(4));
      IF PN -> ON_OFF = '11'B THEN GO TO ERROR;
```

```
/*****      ADD A NODE      *****/

      DEFAULT:
      MES = 'ENTER NEW LABEL...THEN LP ON LABEL';
      CALL UPMES2;
      MES = 'OR ON _SUPER-NODE---';
      CALL UPMES2;
      UNSPEC(CL) = UNSPEC(PN);
      CL = ABS(CL);

      AGAIN2:
      CALL ICURS(G2,CL,NUL,1);
      CALL EXEC(G2);

      ATTENT:

      CALL RQ;
      CALL RCURS(G2);
      IF CODE ¬= 34 THEN DO;
            CALL EXEC(G2);
            RETURN;
            END;

      IF I10(2) = FKEY(7) THEN DO;
            CALL EXEC(G2);
            RETURN;
            END;

      IF I10(2) = FKEY(3) THEN DO;
            /****   NODE ADDED IS TO BE A SUPER-NODE
                     THIS PART OF ROUTINE IS INCOMPLETE      *****/
            END;

      CALL GSPRD(G2,LB,4,1,TERMCODE,CL,NUL);
      IF LB = 'XXXX' THEN DO;
            NDCT = NDCT + 1;
            LB = 'NN'||SUBSTR(CHAR(NDCT),4);
            END;
      IF INDEX(ALPHA,SUBSTR(LB,1,1)) = 0 |
            INDEX(ALPHA,SUBSTR(LB,2,1)) = 0 THEN DO;
            MES = ' INVALID LABEL---TRY AGAIN';
            CALL UPMES2;
            GO TO AGAIN2;
            END;
      CALL STPOS(G2,PN->XCOR - 50.,PN -> YCOR);
      CALL PTEXT(G2,LB,4,CL,NUL,3,PN->XCOR-50.,PN->YCOR);
      IF PN -> ON_OFF = '10'B THEN DO;
            CN = -CL;
```

```
/*****      ADD A NODE      *****/

          CALL INCL(G2,CN);
          END;
          ELSE CALL DISPNOD (PN->XCOR,PN->YCOR,PN);
     PN -> ON_OFF = '11'B;
     CALL ALLOC(1,NN);
     PN -> SNODE = NN;
     NN -> NEXT = TOP -> NEXT;
     TOP -> NEXT = NN;
     NN -> SCREEN = PN;
     NN -> LABEL = LB;

     PUT LIST('NODE '||LB||' ADDED TO GRAPH '||TOP->LABEL)SKIP(2);
     /*   CALL N_USER(NN,1);   */
     RETURN;

     ERROR:
     MES = 'INVALID LP DETECTooTRY AGAIN';
     CALL UPMES2;
     GO TO AGAIN;

     END ADDNODE;
```

```
/*****     DISPLAY AN ARC      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72)');
DISPARC:  PROC(P1,P2,SN);
      DCL (P1,P2,SN)PTR,
      (NUL,CL,G3,KEY3,TOTAL3)FIXED BIN(31)EXT,
      MES CHAR(40) EXT,
      PLINE ENTRY(,,,,,FIXED BIN(31),FIXED BIN(31)),
      (X(4),Y(4),XO,YO,SQ,RX,RY,DX,DY,X1,X2,Y1,Y2,TX,TY)
          FLOAT BIN(21);
      DCL TIMER ENTRY(FIXED BIN);

      DCL   1 NODE BASED(PP),
            2 LABEL CHAR(4),
            2 COUNT FIXED DEC(2),
            2 NEXT PTR,
            2 SNODE PTR,
            2 HINFO PTR,
            2 SCREEN PTR,
            2 USER PTR.
            2 ON_OFF BIT(2),
            2 XCOR FLOAT BIN(21),
            2 YCOR FLOAT BIN(21),
            2 DOWN PTR,
      PP PTR EXT;

      X1 = P1 ->XCOR;
      X2 = P2 -> XCOR;
      Y1 = P1 -> YCOR;
      Y2 = P2 -> YCOR;

      SQ = SQRT((X2-X1)**2 + (Y2-Y1)**2);

      RX = (X2-X1) / SQ;
      RY = (Y2-Y1) / SQ;

      DX = RX * 70.;
      DY = RY * 70.;

      XO = X1  + DX;
      YO = Y1 + DY;
      X(4), X(1) = X2 - DX;
      Y(4), Y(1) = Y2 - DY;


      DX = RX * 35.;
      DY = RY * 35.;
```

```
/*****     DISPLAY AN ARC      *****/

      TX = X(1) - DX;
      TY = Y(1) - DY;


      DX = RY * 15.;
      DY = RX * 15.;

      X(2) = TX + DX;
      X(3) = TX - DX;
      Y(2) = TY - DY;
      Y(3) = TY + DY;

      CL = ABS(BINARY(UNSPEC(SN)));

      CALL STPOS(G3,X0,Y0);
      CALL PLINE(G3,X(1),Y(1),CL ,KEY3,1,4);
      CALL EXEC(G3);
      IF (KEY3/65536 + 40) > TOTAL3 THEN DO;
      MES = 'IMPORTANT...NO MORE ARCS MAY BE';
      CALL UPMES2;
      MES = 'DISPLAYED---BUFFER IS FILLED';
      CALL UPMES2;
      CALL TIMER(2);
      END;
      RETURN;

      END DISPARC;
```

```
     /*****      DISPLAY A NODE      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72)');
DISPNOD :  PROC(X,Y,PN);
      DCL (X,Y) FLOAT BIN(21),
      MES CHAR(40) EXT,
      (G2,CN,KEY2,TOTAL2) FIXED BIN(31) EXT,
      RADIAN FLOAT BIN(21),
      STPOS ENTRY(,FLOAT BIN(21),FLOAT BIN(21)),
      (NX(8),NY(8)) FLOAT BIN(21) EXT,
      PLINE ENTRY(,,,,,FIXED BIN(31),FIXED BIN(31)),
      TIMER ENTRY(FIXED BIN),
      PN PTR;
      UNSPEC(CN) = UNSPEC(PN);
      CN = -ABS(CN);

    /*****   CN....CORRELATION VALUE FOR THE NODE       *****/

      DO I = 1 TO 8;
      RADIAN = FLOAT(I) * .78539;
      NX(I) = X + 70. * COS(RADIAN);
      NY(I) = Y + 70. * SIN(RADIAN);
      END;

      CALL STPOS(G2,X+70.,Y);
      CALL PLINE(G2,NX(1),NY(1),CN,KEY2,1,8);
      CALL EXEC(G2);
      IF (KEY2/65536 + 30) > TOTAL2 THEN DO;
           MES = 'IMPORTANT...NO MORE NODES MAY BE';
           CALL UPMES2;
           MES = 'DISPLAYED---BUFFER IS FILLED ';
           CALL UPMES2;
           CALL TIMER(2);
           END;

      END DISPNOD;
```

```
/*****     CHANGE DISPLAYED POSITION OF A NODE..      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72)');
MOVENOD:  PROC;
      DCL S(2) LABEL,
      I FIXED BIN,
      (PNN,SNN)  PTR,
      (CA,CLN,CNN,CL,CN) FIXED BIN(31);
      DCL (CODE,IIC(10),FKEY(11),G2,NUL,G3) FIXED BIN(31)EXT,
      (FIRST,PS,PN,PF) PTR EXT,
      MES CHAR(40) EXT,
      UPDISP2 ENTRY(PTR,FIXED BIN),
      STPOS ENTRY(,FLOAT BIN(21),FLOAT BIN(21)),
      PTEXT ENTRY(,CHAR(*),FIXED BIN(31),,,FIXED BIN(31),
            FLOAT BIN(21),FLOAT BIN(21)),
      ALLOC ENTRY(FIXED BIN,PTR),
      DISPARC ENTRY,
      TIMER ENTRY(FIXED BIN);
      DCL   1 NODE BASED(PP),
            2 LABEL CHAR(4),
            2 COUNT FIXED DEC(2),
            2 NEXT PTR,
            2 SNODE PTR,
            2 HINFO PTR,
            2 SCREEN PTR,
            2 USER PTR,
            2 ON_OFF BIT(2),
            2 XCOR FLOAT BIN(21),
            2 YCOR FLOAT BIN(21),
            2 DOWN PTR,

      1 ARC BASED (P_ARC),
            2 SON PTR,
            2 SARC PTR,
            2 AINFO PTR,
            2 SCREAN PTR,
            2 USERA PTR,
            2 MORE PTR,
      (PP,P_ARC) PTR EXT;

      PUT LIST('OPTION SELECTED:  MOVE A NODE')SKIP(2);
      CALL UPDISP2(FIRST,2);
      I = 1;

      S(1):
      MES = 'MOVENOD...LP ON NODE TO BE MOVED';
      CALL UPMES2;
      CALL RQ;
```

```
/*****     CHANGE DISPLAYED POSITION OF A NODE...      *****/

     IF CODE ¬= 34 THEN RETURN;

     IF I10(2) = FKEY(7) THEN RETURN;
     IF I10(1) ¬= G2 THEN GO TO E1;
     IF I10(4) = 0 THEN GO TO E1;

     CL = ABS(I10(4));
     CN = -CL;

     S(2):
     MES = 'MOVNODE...LP ON ''XXXX''';
     CALL UPMES2;
     CALL UPDISP2(FIRST,1);
     CALL RQ;
     IF CODE ¬= 34 THEN RETURN;

     IF I10(2) = FKEY(7) THEN RETURN;
     IF I10(1) ¬= G2 THEN GO TO E2;
     IF I10(4) <= 0 THEN GO TO E2;
     CLN = I10(4);
     CNN = -CLN;
     UNSPEC(PNN) = UNSPEC(CLN);
     IF PNN -> ON_OFF = '11'B THEN GO TO E2;
     UNSPEC(PN) = UNSPEC(CL);
     RS = PN -> SNODE;
     CALL STPOS(G2,PN->XCOR-50.,PN->YCOR);
     CALL PTEXT(G2,'XXXX',4,CL,NUL,3,PN->XCOR-50.,PN->YCOR);
     CALL PTEXT(G2,RS->LABEL,4,CLN,NUL,3,PNN->XCOR-50.,PNN->YCOR);
     CALL OMIT(G2,CN);
     CALL INCL(G2,CL);
     IF PNN->ON_OFF = '10'B THEN CALL INCL(G2,CNN);
          ELSE CALL DISPNOD(PNN->XCOR,PNN->YCOR,PNN);
     PNN -> ON_OFF = '11'B;
     PNN -> SNODE = RS;
     PN -> ON_OFF = '10'B;
     RS -> SCREEN = PNN;

     SN = PN -> DOWN;
     DO WHILE(SN¬= NULL);
          IF SN->SARC = NULL THEN GO TO NEXTARC;
          CA = BINARY(UNSPEC(SN));
          CALL OMIT(G3,CA);
          PF = SN -> SON;
          SNN = PNN -> DOWN;
          LOOK:
          IF SNN = NULL THEN GO TO GO_ON;
```

```
/*****     CHANGE DISPLAYED POSITION OF A NODE..      *****/

          DO WHILE(SNN->SON¬= PF);
               SNN = SNN->MORE;
               IF SNN = NULL THEN GO TO GO_ON;
               END;
          GO_ON:
          IF SNN ¬= NULL THEN IF SNN->SARC  ¬= NULL THEN DO;
                              SNN = SNN -> MORE;
                              GO TO LOOK;
                              END;
             ELSE DO;    /*****SNN->SARC  = NULL    *****/
                              CA = BINARY(UNSPEC(SNN));
                              CALL INCL(G3,CA);
                              END;
               ELSE DO:  /*****   SNN = NULL    *****/
                         CALL ALLOC(2,SNN);
                         SNN -> SON = PF;
                         SNN -> MORE = PNN -> DOWN;
                         PNN -> DOWN = SNN;
                         CALL DISPARC(PNN,PF,SNN);
                         END;
          RS = SN -> SARC;
          SNN -> SARC = RS;
          SNN -> AINFO = PNN;
          RS -> SCREAN = SNN;
          SN -> SARC = NULL;
          NEXTARC:
          SN = SN -> MORE;
          END;

     PF = FIRST -> NEXT;
     DO WHILE(PF¬=FIRST);   /*  LOOK FOR ARCS POINTING TO PN  */
          SN = PF -> DOWN;
          DO WHILE(SN¬=NULL);
               IF SN -> SON ¬= PN THEN GO TO END1;
               IF SN -> SARC = NULL THEN GO TO END1;
               CA = ABS(BINARY(UNSPEC(SN)));
               CALL OMIT(G3,CA);
               SNN = PF -> DOWN;
               DO WHILE(SNN¬= NULL);
                    IF SNN->SON =PNN & SNN->SARC=NULL THEN DO:
                         CA = BINARY(UNSPEC(SNN));
                         CALL INCL(G3,CA);
                         GO TO CONT2;   /* UNUSED ORDERS FOUND  */
                         END;
                    SNN = SNN -> MORE;
                    END;
```

```
/*****     CHANGE DISPLAYED POSITION OF A NODE..      *****/

                CALL ALLOC(2,SNN);
                SNN -> SON = PNN;    /**   UNUSED ORDERS NOT FOUND   ***/
                SNN -> MORE = PF -> DOWN;
                PF -> DOWN = SNN;
                CALL DISPARC(PF,PNN,SNN);

                CONT2:
                RS = SN -> SARC;
                SN -> SARC = NULL;
                SNN -> SARC = RS;
                SNN -> AINFO = PF;
                RS -> SCREAN = SNN;
                END1:
                SN = SN -> MORE;
                END;
           PF = PF -> NEXT;
           END;

      PN -> SNODE = NULL;
      RETURN;
      E1:
      MES = 'INVALID LP DETECT---TRY AGAIN';
      CALL UPMES2;
      GO TO S(I);

      E2:
      I = 2;
      GO TO E1;

      END MOVENOD;
```

```
/*****      REMOVE AN ARC      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72)');
RMVARC:  PROC(SM);
      DCL CA FIXED BIN(31),
      (SN,RS,PN,NN,SM,PF)PTR;
      DCL FIRST PTR EXT,
      MES CHAR(40) EXT,
      (CODE,I10(10),FKEY(11),G3) FIXED BIN(31)EXT,
      UPDISP2 ENTRY(PTR,FIXED BIN),
      TIMER ENTRY(FIXED BIN);
      DCL   1 NODE BASED(PP),
            2 LABEL CHAR(4),
            2 COUNT FIXED DEC(2),
            2 NEXT PTR,
            2 SNODE PTR,
            2 HINFO PTR,
            2 SCREEN PTR,
            2 USER PTR,
            2 ON_OFF BIT(2),
            2 XCOR FLOAT BIN(21),
            2 YCOR FLOAT BIN(21),
            2 DOWN PTR,

      1 ARC BASED (P_ARC),
            2 SON PTR,
            2 SARC PTR,
            2 AINFO PTR,
            2 SCREAN PTR,
            2 USERA PTR,
            2 MORE PTR,
      (PP,P_ARC) PTR EXT;
      PUT LIST('OPTION SELECTED:  REMOVE AN ARC')SKIP(2);
      SN = SM;
      IF SN ¬= NULL THEN DO:
            PUT LIST('** BY DEFAULT **');
            CA  = BINARY(UNSPEC(SN));
            GO TO DEFAULT;
            END;

      CALL UPDISP2(FIRST,2);

      AGAIN:
      MES = 'RMVARC...LP ON ARC TO BE REMOVED';
      CALL UPMES2;
      CALL RQ;
      IF CODE ¬= 34 THEN RETURN;
      IF I10(2) = FKEY(7) THEN RETURN;
```

```
/*****      REMOVE AN ARC      *****/

      IF I10(1) ¬= G3 THEN GO TO ERROR1:
      CA = I10(4);
      UNSPEC(SN) = UNSPEC(CA);

      DEFAULT:
      /*    CALL N_USER(SN->SARC,4);    */
      RS = SN -> SON;
      RS -> COUNT = RS -> COUNT - 1;
      PN = SN -> AINFO;
      NN = PN -> SNODE;
      PF = RS->SNODE:
      PUT LIST('ARC REMOVED FROM '||NN->LABEL||
            ' TO '||PF->LABEL)SKIP(2);
      RS = SN -> SARC:
      PF = RS -> SON;
      IF RS -> AINFO ¬= NULL THEN DO;
            CALL RMVSARC(SN):
            RETURN;
            END;
      SN -> SARC = NULL;
      SN = RS;
      CALL OMIT(G3,CA);
      CALL EXEC(G3);

      RS = NN -> DOWN;
      IF RS = NULL THEN DO;
            MES = 'LOGIC ERROR---DUMP  AFTER LP';
            CALL UPMES2;
            CALL RQ;
            RETURN;
            END;
      IF RS = SN THEN DO;
            NN -> DOWN= RS -> MORE;
            GO TO READY:
            END;
      DO WHILE(RS->MORE¬=SN):
            RS = RS -> MORE:
            END;
      RS -> MORE = SN -> MORE;

      READY:
      RS = SN -> SON:
      RS -> COUNT = RS -> COUNT - 1;

      FREE SN -> ARC;
      RETURN;
```

```
/*****      REMOVE AN ARC      *****/

     ERROR1:
     MES = 'INVALID LP DETECT---TRY AGAIN';
     CALL UPMES2;
     CALL TIMER(1);
     GO TO AGAIN;
     END RMVARC;
```

```
/*****      REMOVE A NODE      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72)');
RMVNODE:  PROC;
      DCL (PN     ,NF,NN,RS,SN)PTR;
      DCL (CODE,I10(10),G2,FKEY(11),CL,CN,NUL)FIXED BIN(31)EXT,
      (FIRST,TOP)PTR EXT,
      MES CHAR(40)EXT,
      UPDISP2 ENTRY(PTR,FIXED BIN),
      STPOS  ENTRY(,FLOAT BIN(21),FLOAT BIN(21)),
      PTEXT ENTRY(,CHAR(*),FIXED BIN(31),,,FIXED BIN(31),
            FLOAT BIN(21),FLOAT BIN(21)),
      TIMER ENTRY(FIXED BIN);
      DCL   1 NODE BASED(PP),
            2 LABEL CHAR(4),
            2 COUNT FIXED DEC(2),
            2 NEXT PTR,
            2 SNODE PTR,
            2 HINFO PTR,
            2 SCREEN PTR,
            2 USER PTR,
            2 ON_OFF BIT(2),
            2 XCOR FLOAT BIN(21),
            2 YCOR FLOAT BIN(21),
            2 DOWN PTR,

      1 ARC BASED (P_ARC),
            2 SON PTR,
            2 SARC PTR,
            2 AINFO PTR,
            2 SCREAN PTR,
            2 USERA PTR,
            2 MORE PTR,
      (PP,P_ARC) PTR EXT;

      PUT LIST('OPTION SELECTED:  REMOVE A NODE')SKIP(2);
      CALL UPDISP2(FIRST,2);
      MES = 'RMVNODE...LP ON NODE TO BE REMOVED';
      CALL UPMES2;

      AGAIN:
      CALL RQ;
      IF CODE ¬= 34 THEN RETURN;
      IF I10(1) ¬= G2 THEN GO TO ERROR;
      IF I10(2) = FKEY(7) THEN RETURN;
      IF I10(4) = 0 THEN GO TO ERROR;

      CL = ABS(I10(4));
```

```
/*****     REMOVE A NODE      *****/

      CN = -CL;
      UNSPEC(PN) = UNSPEC(CL);
      NN = PN->SNODE;
      CALL OMIT(G2,CN);
      CALL STPOS(G2,PN->XCOR-50.,PN->YCOR);
      CALL PTEXT(G2,'XXXX',4,CL,NUL,3,PN->XCOR-50.,PN->YCOR);
      PUT LIST('NODE REMOVED:  '||NN->LABEL||' FROM GRAPH '||TOP->LABEL)
          SKIP(2);
      PUT LIST('ALL INCIDENT ARCS WILL NOW BE REMOVED')SKIP(2);
      /*   CALL N_USER(NN,2);   */

      PN -> ON_OFF = '10'B;

    /*****   REMOVE ARCS   *****/

      AGAIN1:
      RS = PN -> DOWN;
      DO WHILE(RS¬=NULL);
      IF RS -> SARC ¬= NULL THEN CALL RMVARC(RS);
                              /*****   POSITIVELY INCIDENT ARCS   *****/
      RS = RS -> MORE;
      END;

      PF = FIRST -> NEXT;
      DO WHILE(PF¬=FIRST);
            SN = PF -> DOWN;
            DO WHILE(SN¬=NULL);
                  IF SN -> SON = PN THEN IF SN -> SARC ¬= NULL THEN
                                       /*****   NEGATIVELY INCIDENT ARCS  **/
                        CALL RMVARC(SN);
                  SN = SN -> MORE;
                  END;
            PF = PF -> NEXT;
            END;
      NF = TOP;
      NN = TOP -> NEXT;
      DO WHILE(NN¬=TOP);
            IF NN -> SCREEN = PN THEN GO TO CONT;
            NF = NN;
            NN = NN -> NEXT;
            END;
    /*****   ERROR  IN  LOGIC            *****/
      MES = 'LOGIC ERROR---DUMP  AFTER LP';
      CALL UPMES2;
      CALL RQ;
      RETURN;
```

```
/*****      REMOVE A NODE      *****/

      CONT:
      NF -> NEXT = NN -> NEXT;
      IF NN -> COUNT ¬= 0 | NN -> DOWN ¬= NULL THEN
           PUT LIST('***NN PARMS WRONG:  NN = '||DECIMAL(UNSPEC(NN))||
                ' COUNT = '||NN->COUNT||' DOWN = '||
                DECIMAL(UNSPEC(NN->DOWN)))SKIP(2);

      IF NN -> SNODE ¬= NULL THEN DO;
           CALL RMVSNOD(NN);
           RETURN;
           END;

      FREE NN -> NODE;
      PN -> SNODE = NULL;

      RETURN;

      ERROR:
      MES = 'INVALID LP DETECT---TRY AGAIN';
      CALL UPMES2;
      GO TO AGAIN;

      END RMVNODE;
```

```
/*****   REQUEST ATTENTION INFORMATION   *****/

  * PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72)');
  RQ:   PROC:
     DCL (ATTEN,CODE,I10(10),FKEY(11)) FIXED BIN(31) EXT,
       TOP PTR EXT,
       FRAME CHAR(26) EXT,
       CLN ENTRY(CHAR(*)) RETURNS(CHAR(132)VAR),
       RQATN ENTRY(,,FIXED BIN(31),,FIXED BIN(31),FIXED BIN(31),
            FIXED BIN(31));
       DCL   1 NODE BASED(PP),
             2 LABEL CHAR(4),
             2 COUNT FIXED DEC(2),
             2 NEXT PTR,
             2 SNODE PTR,
             2 HINFO PTR,
             2 SCREEN PTR,
             2 USER PTR,
             2 ON_OFF BIT(2),
             2 XCOR FLOAT BIN(21),
             2 YCOR FLOAT BIN(21),
             2 DOWN PTR,
       PP PTR EXT;
       CALL RQATN(ATTEN,CODE,2,I10(1),0,-31,34);
       PUT LIST(REPEAT('*',49))SKIP(5);
       PUT LIST('INTERRUPT SOURCE:')SKIP(2);
       IF CODE = -1 THEN CODE = 0;
       IF CODE = 34 THEN DO;
            PUT LIST('LIGHT PEN');
            END;

            ELSE DO;
            PUT LIST(CLN('FUNCTION KEY NUMBER '||CODE));
            END;
       PUT LIST('FRAME DISPLAYED:  '||FRAME)SKIP;
       IF FRAME = 'GRAPH MANIPULATION FRAME' THEN IF TOP = NULL THEN
            PUT LIST('GRAPH DISPLAYED:'||REPEAT(' ',8)||'(NONE)')SKIP;
       ELSE PUT LIST('GRAPH DISPLAYED:'||REPEAT(' ',8)||'('||TOP->LABEL||
            ')')SKIP;
       IF CODE = 34 THEN IF I10(2) = FKEY(7) THEN
            PUT LIST('OPTION SELECTED:  CANCEL')SKIP(2);

       END RQ;
```

```
/*****     UPDATE OPERATOR MESSAGE     *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72)');
UPMES2:  PROC:
      DCL (MES2,NUL,KM2,KM22) FIXED BIN(31)EXT,
      MES CHAR(40) EXT,
      STPOS ENTRY(,FLOAT BIN(21),FLOAT BIN(21)),
      PTEXT ENTRY(,CHAR(*),FIXED BIN(31),,,FIXED BIN(31),
           FLOAT BIN(21),FLOAT BIN(21));

      IF INDEX(MES,'ooo') ¬= 0 THEN DO;
      CALL STPOS(MES2,1o,3o);
      CALL PTEXT(MES2,MES,LENGTH(MES),NUL,KM2,3,1o,3o);
      CALL PTEXT(MES2,REPEAT(' ',39),40,NUL,KM22,3,1o,5o);
      END;

      ELSE IF INDEX(MES,'---') ¬= 0 THEN DO;
      CALL STPOS(MES2,1o,5o);
      CALL PTEXT(MES2,MES,LENGTH(MES),NUL,KM22,3,1o,5o);
      END;

      ELSE DO;
      CALL STPOS(MES2,1o,3o);
      CALL PTEXT(MES2,MES,LENGTH(MES),NUL,KM2,3,1o,3o);
      END;
      CALL EXEC(MES2);
      PUT LIST('  MESSAGE DISPLAYED:  '||MES)SKIP(2);

      END UPMES2;
```

```
/*****      ADD AN ARC TO THE BOTTOM OF A LIST      *****/

  * PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72),D');
  ADARCBT:   PROC(PN,SN);
        DCL   1 NODE BASED(PP),
              2 LABEL CHAR(4),
              2 COUNT FIXED DEC(2),
              2 NEXT PTR,
              2 SNODE PTR,
              2 HINFO PTR,
              2 SCREEN PTR,
              2 USER PTR,
              2 ON_OFF BIT(2),
              2 XCOR FLOAT BIN(21),
              2 YCOR FLOAT BIN(21),
              2 DOWN PTR,

        1 ARC BASED (P_ARC),
              2 SON PTR,
              2 SARC PTR,
              2 AINFO PTR,
              2 SCREAN PTR,
              2 USERA PTR,
              2 MORE PTR,
        (PP,P_ARC) PTR EXT;
        DCL (PN,PF,SN) PTR;
        PF = PN -> DOWN;
        IF PF = NULL THEN DO;
              PN -> DOWN = SN;
              RETURN;
              END;
        DO WHILE(PF->MORE ¬= NULL);
              PF = PF -> MORE;
              END;
        PF -> MORE = SN;
        END ADARCBT;
```

```
/*****     ALLOCATE EITHER A HEADER OR ARC CELL      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72),D');
ALLOC:  PROC(TYPE,POINTER);
      DCL   1 NODE BASED(PP),
              2 LABEL CHAR(4),
              2 COUNT FIXED DEC(2),
              2 NEXT PTR,
              2 SNODE PTR,
              2 HINFO PTR,
              2 SCREEN PTR,
              2 USER PTR,
              2 ON_OFF BIT(2),
              2 XCOR FLOAT BIN(21),
              2 YCOR FLOAT BIN(21),
              2 DOWN PTR,

      1 ARC BASED (P_ARC),
              2 SON PTR,
              2 SARC PTR,
              2 AINFO PTR,
              2 SCREAN PTR,
              2 USERA PTR,
              2 MORE PTR,
      (PP,P_ARC) PTR EXT;
      DCL TYPE FIXED BIN,
      POINTER PTR;

      IF TYPE ¬= 1 & TYPE ¬= 2 THEN
      PUT LIST('CALL TO ALLOC...TYPE = '||TYPE)SKIP(1);
      IF TYPE = 1 THEN DO;

    /*****   ALLOCATE AND INITIALIZE HEADER CELL   *****/
      ALLOCATE  NODE  SET(POINTER);
      POINTER -> LABEL = '    ';
      POINTER -> XCOR,  POINTER -> YCOR, POINTER -> COUNT = 0,;
      POINTER -> NEXT,  POINTER -> SNODE,
            POINTER -> HINFO, POINTER -> DOWN,
      POINTER -> SCREEN,  POINTER -> USER = NULL;
      POINTER -> ON_OFF = '00'B;
      RETURN:
      END;

      IF TYPE = 2 THEN DO;

    /*****   ALLOCATE AND INITIALIZE ARC CELL   *****/
      ALLOCATE ARC SET(POINTER);
      POINTER -> SON,  POINTER -> SARC,
```

```
/*****      ALLOCATE EITHER A HEADER OR ARC CELL      *****/

          POINTER -> AINFO,  POINTER -> MORE,
          POINTER -> SCREAN, POINTER -> USERA = NULL;
     RETURN;
     END;
     END ALLOC;
```

```
      /*****     CLEAN UP PRINTER LINE      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72),D');
CLEANUP:   PROC(OUT);
      DCL OUT CHAR(*) VAR;
      DCL (LAST,I,J) FIXED BIN;

      LAST = LENGTH(OUT);
      DO I = LAST TO 1 BY -1;
      IF SUBSTR(OUT,I,1) ¬= ' ' THEN DO;
           OUT = SUBSTR(OUT,1,I);
           GO TO L1;
           END;
      END;

      L0:
      J = INDEX(OUT,' (');
      IF J ¬= 0
           THEN DO;
                OUT = SUBSTR(OUT,1,J-1)||SUBSTR(OUT,J+1);
                GO TO L0;
                END;

      L2:
      J = INDEX(OUT,'( ');
      IF J ¬= 0
           THEN DO;
                OUT = SUBSTR(OUT,1,J)||SUBSTR(OUT,J+2);
                GO TO L2;
                END;

      L1:
      K = INDEX(OUT,' ,');
      J = INDEX(OUT,'  ');
      IF J = 0 THEN IF K = 0 THEN RETURN;
           ELSE OUT = SUBSTR(OUT,1,K-1)||SUBSTR(OUT,K+1);
      ELSE OUT = SUBSTR(OUT,1,J) || SUBSTR(OUT,J+2);
      GO TO L1;
      END CLEANUP;
```

```
/*****     FUNCTION TO CLEAN UP PRINTED LINE     *****/

* PROCESS('A,X,NT,E,SIZE=999990,SM=(1,72),D');
CLN:   PROC(IN) CHAR(132)VAR;
       DCL  IN CHAR(*),
       OUT CHAR(132)VAR;
       OUT = IN;
       CALL CLEANUP(OUT);
       RETURN(OUT);
       END CLN;
```

```
/*****      INITIALIZE GRAPHIC DATA SET G1      *****/

  * PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72),D');
  INITG1:  PROC;
        DCL  (G1,NUL,IC(3),CHOICE)FIXED BIN(31) EXT,
        PTEXT ENTRY(,,FIXED BIN(31),,,FIXED BIN(31),FLOAT BIN(21),FLOAT BIN
        (21)),
        SCHAM ENTRY(,FIXED BIN(31)),
        SDATL ENTRY(,FLOAT BIN(21),FLOAT BIN(21),FLOAT BIN(21),FLOAT BIN
              (21)),
        SGDSL ENTRY(,FLOAT BIN(21),FLOAT BIN(21),FLOAT BIN(21),
                    FLOAT BIN(21),FLOAT BIN(21),FLOAT BIN(21),
                    FLOAT BIN(21),FLOAT BIN(21)),
        STPOS ENTRY(,FLOAT BIN(21),FLOAT BIN(21));
        CALL SGDSL(G1,6.5,12.,42.5,23.,0.,0.,49.,35.);
        CALL SDATL(G1,.5,11.5,36.5,.5);
        CALL SCHAM(G1,2);
        CALL STPOS(G1,14.,1.);
        CALL PTEXT(G1,'I G R A P',9,NUL.NUL,NUL,14.,1.);
        CALL PTEXT(G1,REPEAT('=',10),11,NUL,NUL,NUL,13.,2.);
        CALL PTEXT(G1,'INTERACTIVE',11,NUL,NUL,NUL,13.,3.);
        CALL PTEXT(G1,'GRAPH REDUCTION AND ANALYSIS PROGRAM',
              36,NUL,NUL,NUL,1.,4.);
        CALL PTEXT(G1,'INPUT OPTIONS:',14,NUL,NUL,NUL,9.,6.);
        CALL PTEXT(G1,'1.  BOOB TUBE',13,IC(1),NUL ,NUL, 9.,7.);
        CALL PTEXT(G1,'2.  CARDS',9,IC(2),NUL,NUL, 9.,8.);
        CALL PTEXT(G1,'3.  MAGNETIC DEVICE',19,IC(3),NUL,NUL,9.,9.);
        CALL PTEXT(G1,'** SELECT ONE OF THE OPTIONS **',31,NUL,NUL,NUL,
                  4.,11.);
        CALL STPOS(G1,7.,7.);
        CALL PTEXT(G1,' ',1,NUL,CHOICE,NUL,7.,7.);
        END INITG1;
```

```
/*****      PRINT GRAPH STRUCTURE      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72),D');
P_ARCS:  PROC(P,ID);
     DCL (P,PT,SON1,RLSN) PTR,
     OUT CHAR(132)VAR,
     (NON,NOA) FIXED DEC(3),
     (NOSK,NOSC) FIXED DEC(3),
     START CHAR(4),
     ID CHAR( *);
     DCL   1 NODE BASED(PP),
           2 LABEL CHAR(4),
           2 COUNT FIXED DEC(2),
           2 NEXT PTR,
           2 SNODE PTR,
           2 HINFO PTR,
           2 SCREEN PTR,
           2 USER PTR,
           2 ON_OFF BIT(2),
           2 XCOR FLOAT BIN(21),
           2 YCOR FLOAT BIN(21),
           2 DOWN PTR,
     1 ARC BASED (P_ARC),
           2 SON PTR,
           2 SARC PTR,
           2 AINFO PTR,
           2 SCREAN PTR,
           2 USERA PTR,
           2 MORE PTR,
     (PP,P_ARC) PTR EXT:
     NON, NOA = 0;
     NOSC, NOSK = 0;
     PUT LIST('SYSTEM ROUTINE P_ARCS INVOKED')SKIP(2);

     IF P = NULL THEN DO;
          PUT LIST('TOP IS NULL')SKIP(2):
          RETURN;
          END;

     OUT =     '  GRAPH NAME:  '||P->LABEL||'('||P->COUNT||'):  '||ID ;
     CALL CLEANUP (OUT):
     PUT LIST(OUT)SKIP(4);

     IF P->NEXT = P THEN DO;
          PUT LIST('GRAPH HAS NO NODES')SKIP(2);
          RETURN;
          END;
```

```
/*****     PRINT GRAPH STRUCTURE     *****/

     PT = P;
     START = P->LABEL;

     FATHER:
     PT = PT -> NEXT;
     IF PT -> LABEL = START
          THEN DO;
          OUT =     'THERE ARE '||NOSC||' SOURCE(S), '||NOSK||' SINK(S),
'||NON||' NODE(S), AND '||NCA||' ARC(S).';
          CALL CLEANUP (OUT);
          PUT LIST(OUT)SKIP(3);
          RETURN;
          END;

     OUT = PT->LABEL||DECIMAL(UNSPEC(PT))||'('||PT->COUNT||') -> ';
     IF PT -> SNODE ¬= NULL THEN DO;
          OUT = SUBSTR(OUT,1,INDEX(OUT,')')-1)||',SUPER NODE'
            ||SUBSTR(OUT,INDEX(OUT,')'));
          END;
     IF PT -> COUNT = 0
          THEN DO;
          NOSC = NOSC + 1;
          OUT = 'SOURCE:  '||OUT;
          END;
     NON = NON + 1;

     SON1 = PT -> DOWN;
     ICT = 0;
     GO TO SON_CONTINUE;

     SONS:
     SON1 = SON1 -> MORE;

     SON_CONTINUE:
     IF SON1 = NULL THEN DO;
          IF ICT = 0 THEN DO;
               OUT = OUT||'SINK';
               NOSK = NOSK + 1;
               END;
          CALL CLEANUP (OUT);
          PUT LIST('  '||OUT)SKIP(2);
          GO TO FATHER;
          END;

     ICT = ICT +1;
     NOA = NOA + 1;
```

```
/*****      PRINT GRAPH STRUCTURE      *****/

      RLSN = SON1 -> SON;
      IF ICT ¬= 1 THEN OUT = OUT||', ';
      OUT = OUT||RLSN->LABEL||DECIMAL(UNSPEC(RLSN));
      IF SON1 -> AINFO ¬= NULL
           THEN OUT = OUT||'(SUPER ARC)';

      GO TO SONS;
      END P_ARCS;
```

```
/*****      REMOVE SUPERARC      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72),D');
RMVSARC:  PROC(SN);
      DCL  SN PTR;
      DCL   1 NODE BASED(PP),
              2 LABEL CHAR(4),
              2 COUNT FIXED DEC(2),
              2 NEXT PTR,
              2 SNODE PTR,
              2 HINFO PTR,
              2 SCREEN PTR,
              2 USER PTR,
              2 ON_OFF BIT(2),
              2 XCOR FLOAT BIN(21),
              2 YCOR FLOAT BIN(21),
              2 DOWN PTR,
      PP PTR EXT;
      PUT LIST('****RMVSARC DUMMY CALLED...'||
            'SN = '||DECIMAL(UNSPEC(SN)))SKIP(2);
      /*****   THIS ROUTINE IS NOT COMPLETE      *****/
      RETURN;
      END RMVSARC;
```

```
/*****      REMOVE A SUPERNODE      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72),D');
RMVSNOD:  PROC(NN);
      DCL NN PTR;
      DCL   1 NODE BASED(PP),
            2 LABEL CHAR(4),
            2 COUNT FIXED DEC(2),
            2 NEXT PTR,
            2 SNODE PTR,
            2 HINFO PTR,
            2 SCREEN PTR,
            2 USER PTR,
            2 ON_OFF BIT(2),
            2 XCOR FLOAT BIN(21),
            2 YCOR FLOAT BIN(21),
            2 DOWN PTR,
      PP PTR EXT;
      DCL   1 ARC BASED (P_ARC),
            2 SON PTR,
            2 SARC PTR,
            2 AINFO PTR,
            2 SCREAN PTR,
            2 USERA PTR,
            2 MORE PTR,
      P_ARC PTR EXT;
     PUT LIST('***RMVSNOD DUMMY CALLED...NN = '||DECIMAL(UNSPEC(NN))||
               NN -> LABEL)SKIP(2);
      RETURN;
      END RMVSNOD;
```

```
/*****      TIMER      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72),D');
TIMER:   PROC(I);
      DCL (I,ETIM) FIXED BIN,
      TIM CHAR(9) EXT;
      TIM = TIME;
      ETIM = BINARY(SUBSTR(TIM ,5,2)) + I + 1;
      IF ETIM > 60 THEN ETIM = ETIM - 60;
      DO WHILE('1'B);
           TIM = TIME;
           IF ETIM = BINARY(SUBSTR(TIM ,5,2)) THEN RETURN;
      END TIMER;
```

```
/*****     REMOVE OR DISPLAY AVAILABLE NODE POSITIONS      *****/

* PROCESS('A,X,NT,E,SIZE=999999,SM=(1,72),D');
UPDISP2:  PROC(TOP,I);
      DCL   1 NODE BASED(PP),
              2 LABEL CHAR(4),
              2 COUNT FIXED DEC(2),
              2 NEXT PTR,
              2 SNODE PTR,
              2 HINFO PTR,
              2 SCREEN PTR,
              2 USER PTR,
              2 ON_OFF BIT(2),
              2 XCOR FLOAT BIN(21),
              2 YCOR FLOAT BIN(21),
              2 DOWN PTR,
      PP PTR EXT;
      DCL TOP PTR,
      ISW BIT(1) EXT,
      PN PTR EXT,
      (CL,G2) FIXED BIN(31) EXT,
      I FIXED BIN;

      IF TOP = NULL THEN RETURN;
      PN = TOP -> NEXT;
      IF I ¬= 1 THEN GO TO TWO;

      IF ISW THEN RETURN;
      DO WHILE (PN¬=TOP);
      IF ¬SUBSTR(PN->ON_OFF,2,1) THEN DO;
      UNSPEC(CL) = UNSPEC(PN);
      CL = ABS(CL);
      CALL INCL(G2,CL);
      END;
      PN = PN -> NEXT;
      END;
      ISW = '1'B;
      CALL EXEC(G2);
      RETURN;

      TWO:
      IF ¬ISW THEN RETURN;
      DO WHILE (PN¬=TOP);
      IF ¬SUBSTR(PN->ON_OFF,2,1) THEN DO;
      UNSPEC(CL) = UNSPEC(PN);
      CL = ABS(CL);
      CALL OMIT(G2,CL);
      END;
```

```
/*****     REMOVE OR DISPLAY AVAILABLE NODE POSITIONS      *****/
      PN = PN -> NEXT;
      END;
      ISW = '0'B;
      CALL EXEC(G2);
      END UPDISP2;
```

BIBLIOGRAPHY

1. Busacker, Robert G. and Saaty, Thomas L., Finite Graphs and Networks, McGraw-Hill, 1965.

2. Crespi-Reghizzi, S. and Morpurgo, R., "A Language for Treating Graphs," Communications of the ACM, v. 13, p. 319-323, May 1970.

3. Wolfberg, Michael S., "An Interactive Graph Theory System," Computing Reviews, v. 11, p. 167, March 1970.

4. Holifield, Claude, Graphic Display and Manipulation of Tree-type Data Structures, Masters Thesis, United States Naval Postgraduate School, Monterey, California, 1969.

5. International Business Machines, Inc., IBM System/360 Operating System Graphic Subroutine Package (GSP) for FORTRAN IV, COBOL, and PL/I, Form C27-6932, 4th ed., 1969.

6. International Business Machines, Inc., IBM System/360 Components Description IBM 2250 Display Unit Model 1, Form A27-2701, 3rd ed., January 1969.

7. International Business Machines, Inc., IBM System/360 Operating System PL/I (F) Language Reference Manual, Form C28-8201, 3rd ed., October 1969.

8. Knuth, Donald E., Fundamental Algorithms, Addison Wesley, 1968.

9. Licklider, J. C. R., "Graphic Input--Survey of Techniques," Computer Graphics, Fred Gruenberger ed., p. 39-69, Thompson Book Co., 1967.

10. Licklider, J. C. R., "Man-Computer Symbiosis," IRE Transactions on Human Factors in Electronics, v. 1, p. 4-11, March 1960.

11. Withington, Frederic G., "Cosmetic Programming," Datamation, v. 16, p. 91-95, March 1970.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Associate Professor U. R. Kodres Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	LTJG James W. Thomas, USN Class 70-3 NAVNUPWRSCOL Mare Island, California 94592	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY (Corporate author)

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

REPORT TITLE

Interactive Graph Reduction and Analysis Program

DESCRIPTIVE NOTES (Type of report and inclusive dates)

Master's Thesis; June 1970

AUTHOR(S) (First name, middle initial, last name)

James Winton Thomas

REPORT DATE

June 1970

7a. TOTAL NO. OF PAGES

135

7b. NO. OF REFS

11

CONTRACT OR GRANT NO.

PROJECT NO.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

DISTRIBUTION STATEMENT

This document has been approved for public releas and sale; its distribution is unlimited.

SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

Naval Postgraduate School
Monterey, California 93940

ABSTRACT

This paper describes the design of an interactive system to aid in the analysis of problems which involve directed graphs. The digital computing system is assumed to have a graphic display device on which directed graphs may be drawn and from which light pen, function keyboard, and alphanumeric keyboard information may be transmitted on-line to the system. Directed graphs are represented in core storage by a dynamically allocated hierarchical list structure. User-written analysis routines are linked to the system to apply it to a particular field of problems. An initial implementation of its capabilities on the IBM 360/67 with an IBM 2250 Display Unit was written in PL/I (F). Under the IBM System/360 Operating System, it executed in less than 200K bytes and provided reasonable response to on-line interaction.

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Directed graph						
Graph theory						
Interactive						
Graphic						
Display